1989

VLSI implementation of distributed arithmetic

Dajen Huang
Lehigh University

VLSI IMPLEMENTATION OF DISTRIBUTED ARITHMETIC

by

Dajen Huang

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the degree of
Master of Science
in
Electrical Engineering

Lehigh University
1989

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

April 26, 1989
Date

Advisor in Charge

April 26, 1989
Date

CSEE Department Chairperson

TABLE OF CONTENTS

CHAPTER 5

A DISCRETE COSINE TRANSFORM CHIP

CHAPTER 6

CONCLUSION

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The advancement of microelectronic technology and new algorithm development are dependent of each other. Current VLSI technology open the way to implement digital signal processing(DSP) chip for those applications involving large signal bandwidths and high speed computation. When the fundamental operations of convolution and multiplication are mixed—distributed arithmetic (contrary to the classical "concentrated arithmetic") is very attractive in DSP hardware realizations due to its compatibility and flexibility.

A 4-tap convolution processing unit is designed and fabricated in double-metal 2μm CMOS technology. Its 33 MHz operating frequency equivalently complete 99 millions of adds and 132 millions of multiplys in a second. This characterized 4-tap unit could be easily expanded to larger tap sizes(32, for example) and higher precision but only limited by the available silicon area. By doing specific permutations on input data sequence, Discrete Cosine Transform(DCT) becomes as circular correlation. Distributed arithmetic is also efficient in this computation. A DCT chip employed in image coding, polyphase filter banks, and FFT evaluation is investigated and ready for VLSI implementation. Distributed Arithmetic is therefore shown as an efficient algorithm in VLSI implementations of digital filter(linear convolution), DFT(circular convolution), as well as DCT(circular correlation).

CHAPTER 1

INTRODUCTION

The aim of this study is to investigate VLSI (Very Large Scale Integrated circuit) implementation of "distributed arithmetic" for DSP (digital signal processing) applications. The vehicle chosen is the fundamental building block in digital system design—digital filter, or say a convolution processor. To exhibit the feasibility of this algorithm in different operation, a DCT (discrete cosine transform) chip was also derived as another example.

The techniques and applications of DSP field are as old as Newton and Gauss and as new as digital computers and integrated circuits. During 1950s, the use of digital computers in signal processing arose because the flexibility of digital computers was useful to simulate a signal processing system before implementing it in analog hardware. In this way, a new signal processing algorithm, or system, could be studied in a flexible experimental environment before committing economic and engineering resources to constructing. However, the advent of modern semiconductor technology has caused a revolution that more and more implementations of digital algorithms will be in terms of special purpose hardware rather than as software for a general purpose computer. Furthermore, some sophisticated signal processing algorithms which previously had appeared to be impractical began to appear to have pratical implementations with the

current VLSI technology. This also explains the current interest in new digital signal processing algorithms for VLSI.

In this study, the mathematical description and design of a 4-tap convolution unit and a 32×32 DCT chip are discussed. The design procedure is described in detail and involved using many CAD tools; SPICE (Simulation Program with Integrated Circuit Emphasis) [1] for circuit simulation, BSIM (Behavior SIMulation) [2], RSIM [3] for logic/timing simulation, NET [4] for hardware description, and MAGIC [5] for layout, etc. The use of the appropriate computer simulations and checks enables first pass IC design work.

The idea of distributed arithmetic is outlined in the next chapter. It shows how linear convolution (digital filter), and circular correlation(DCT) is possibly realized without using multiplier. The DCT example requires a permutation on the input data sequence before we can treat it as a cyclic convolution, while the linear convolution is obvious. Several different hardware mechanisms for distributed arithmetic implementation, namely, the trade-off between arithmetic operation and silicon area are discussed too.

Hardware realization—the path from abstract algorithm to real chip is examined in chapter 3. The chip structure and leaf-cell design are crucial in determination of chip speed, thus they are analyzed in detail. The sign bit problem in dealing with arithmetic

operations of 2's complement numbers is coved.

In chapter 4, the design procedure and CAD tool used through the whole project are presented. A multiplexed serial-out and a single phase speed testing methods are considered to make the chip testable under pipelined architecture. Characterizations of the fabricated convolution chip are described, including functionality and speed performance.

The well simulated DCT chip, based on a new developed algorithm, is given in chapter 5. Its behavior model and simulation results are enclosed in the appendix for reference.

CHAPTER 2

DISTRIBUTED ARITHMETIC

2.1 CONVENTIONAL DESCRIPTIONS OF DIGITAL FILTER

In the 1980s, VLSI developments have dramatically reduced the cost and power consumption of digital filters and have led to much more widespread application of digital signal processing.

Digital filter is superior to its analog counterpart in the following aspects:

1. Programmable (filter characteristics easily changed)

2. Reliable and repeatable

3. Free from component drift

4. No tuning required

5. No precision components, no component matching

6. Superior performance (linear phase, for example)

Different mathematical descriptions of digital filter may suggest different hardware realizations. After a brief review of general digital filter forms, the way of using distributed arithmetic in digital filter implementation is discussed.

The transfer characteristics of a digital filter are commonly described in terms of

its Z-domain transfer function [6],

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{i=0}^{N} b_i z^{-i}}{1 + \sum_{i=0}^{N} a_i z^{-i}}, \tag{2.1}$$

where z^{-1} represents the unit delay. The corresponding difference equation is,

$$y(k) = \sum_{i=0}^{N} b_i \cdot x(k\text{-}i) - \sum_{i=0}^{N} a_i \cdot y(k\text{-}i), \tag{2.2}$$

then the directly equivalent digital circuit form can be realized. But there are three canonical forms, or variations thereof, are most often employed. These forms are canonical in the sense that a minimum number of adders, multipliers, and delays (shift register) are required to realize (2.1) in the general case. The first of these forms reduces the number of delays to N by considering ,

$$H(z) = H_1(z) \cdot H_2(z) = \frac{W(z)}{X(z)} \cdot \frac{Y(z)}{W(z)} = \left(\frac{1}{1 + \sum_{i=0}^{N} a_i z^{-i}} \right) \cdot \left(\sum_{i=0}^{N} b_i z^{-i} \right). \tag{2.3}$$

Or, in terms of difference equations:

$$w(k) = x(k) - \sum_{i=0}^{N} a_i w(k\text{-}i), \text{ and}$$

$$y(k) = \sum_{i=0}^{N} b_i w(k\text{-}i). \tag{2.4}$$

However, it has been pointed out by [7], [8] that use of this direct form is avoided because the accuracy requirements on the coefficients $\{a_i\}$ and $\{b_i\}$ are often severe (finite register effect). The second canonical form corresponding to a factorization of the numerator and denominator polynomials of (2.1) to produce:

$$H(z) = c \cdot \prod_{i=0}^{m} H_i(z) = c \cdot \prod_{i=0}^{m} \frac{b_{2i}z^{-2} + b_{1i}z^{-1} + 1}{a_{2i}z^{-2} + a_{1i}z^{-1} + 1}, \tag{2.5}$$

where m is the integer part of (n+1)/2. This is the cascade form of a digital filter. Note a_{2i} and b_{2i} could be zero for some i. The third canonical form resulted from a partial fraction expansion of (2.1), such that:

$$H(z) = c + \sum_{i=0}^{m} H_i(z) = c + \sum_{i=0}^{m} \frac{b_{1i}z^{-1} + b_{0i}}{a_{2i}z^{-2} + a_{1i}z^{-1} + 1}, \tag{2.6}$$

where $c=b_n/a_n$. This realization is called a parallel form.

In summary, all three canonical forms are entirely equivalent with regard to the amount of storage required (N shift registers) and the number of arithmetic operations required (2N+1 multipliers and 2N adders). Next, we will see how distributed arithmetic is applied to digital filter implementations.

2.2 DISTRIBUTED ARITHMETIC

Distributed arithmetic was suggested in middle 70's as a new hardware realization [9] [10] which was based on the fact that all the coefficients of a digital filter are constant so that it is possible to use ROM (Read Only Memory) instead of multipliers. Later on, Burrus [11] provided a mathematical framework for this new approach to digital filter implementations.

It has been noticed that multidimensional convolution techniques when used with various fast algorithms for short length convolutions, will improve the efficiency of one-dimensional convolution of the original signals [12]. The basic idea of distributed arithmetic is to observe that conventional convolution is already two dimensional.

$$y(k) = \sum_{i=0}^{N-1} h(i)\cdot x(k-i) = \sum_{i=0}^{N-1} \left(\sum_{b_1=0}^{B_h-1} h(i,b_1)\cdot 2^{-b_1} \right) \left(\sum_{b_2=0}^{B_x-1} x(k-i,b_2)\cdot 2^{-b_2} \right)$$

$$= \sum_{i=0}^{N-1} \sum_{b_1=0}^{B_h-1} \sum_{b_2=0}^{B_x-1} h(i,b_1)\cdot x(k-i,b_2)\cdot 2^{-(b_1+b_2)}, \qquad (2.7)$$

where, $h(i,b_1)$ and $x(k-i,b_2) \in \{0,-1\}$, if b1=0 or b2=0,

$\in \{0,1\}$, else. (2.8)

Changing index with $m=b_1+b_2$, $M=B_h+B_x-1$ gives

$$y(k) = \sum_{m=0}^{M-1} y_0(k, m)\cdot 2^{-m} = \sum_{m=0}^{M-1}\sum_{i=0}^{N-1}\sum_{b_1=0}^{B_h-1} h(i,b_1)\cdot x(k-i,m-b_1)\cdot 2^{-m}, \qquad (2.9)$$

This is a standard two-dimensional convolution form. Data sequence here is viewed as a two-dimensional binary (again, note $h(\cdot)$ and $x(\cdot) \in \{0,1\}$) signal with the rows giving the word and the columns giving the binary position. Thus one-dimensional numbers string convolution become two-dimensional bits array convolution. For example;

1−D: Let, $x(k)*h(k) = \{x(0), x(1)\} * \{h(0), h(1)\}$

$$= \{x(0)\cdot h(0),\ x(0)\cdot h(1)+x(1)\cdot h(0),\ x(1)\cdot h(1)\} = y(k) \qquad (2.10)$$

2-D: Assume all word's are two bit long, then the output y is three bit long.

$$\begin{bmatrix} x_{0,0} & x_{0,1} \\ x_{1,0} & x_{1,1} \end{bmatrix} * \begin{bmatrix} h_{0,0} & h_{0,1} \\ h_{1,0} & h_{1,1} \end{bmatrix} = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} \\ y_{1,0} & y_{1,1} & y_{1,2} \\ y_{2,0} & y_{2,1} & y_{2,2} \end{bmatrix} \qquad (2.11)$$

Now, operation along the horizontal dimension is the convolution (words multiplication is bits convolution),

Ex: $y(0) = x(0)\cdot h(0) = [x_{0,0}\ \ x_{0,1}] * [h_{0,0}\ \ h_{0,1}]$.

and the operation along the vertical dimension is the usual multiplication,

Ex: $y(k,0) = [x_{0,0}\cdot h_{0,0}\ \ x_{0,0}\cdot h_{1,0}+x_{1,0}\cdot h_{0,0}\ \ x_{1,0}\cdot h_{1,0}]$.

Down to bit level, arithmetic operations can be "re-distributed" because the order of those two operations, multiplication and convolution, are interchangeble—hence the name "distributed arithmetic". Furthermore, if we convert (2.9) into a one-dimensional bitstring-to-bitstring convolution, several interesting mechanisms can be developed by using different block length [10]. One of the extreme cases is the convolution of a word string with a bit string,

$$\boxed{y(k) = \sum_{b_1=0}^{B_x-1} y_f(k, b_1)\cdot 2^{-b_1} = \sum_{b_1=0}^{B_x-1} \sum_{i=0}^{N-1} h(i)\cdot x(k-i, b_1)\cdot 2^{-b_1}} \qquad (2.12)$$

It is the scheme used through this paper. Note $x(\cdot)$ sequence is "distributed" in the arithmetic operation. It is very attractive because table-look-up is possibly used to pre-calculate the partial products, $\sum_{i=0}^{N-1} h(i)\cdot x(k-i,\ b_1)$, which was previously done by sophisticated multiplications. The detailed hardware realization will be discussed later.

2.3 DIGITAL FILTER STRUCTURES DESCRIBED BY DISTRIBUTED ARITHMETIC

It is the implementation of digital filters with all precalculated partial products stored in memory that received the greatest interest. Observing (2.12) the B_x's partial products $y_f(k,b_1)$ are nothing but the possible summations of filter's coefficients $h(i)$ due to the fact of $x(k-i,b_1) \in \{0,1\}$. Therefore all 2^N partial products could be precalculated

and stored in a memory, then the input data sequence $\{x(k)\}$ is used to address out $y_f(k,b_1)$ for $b_1 = [0, B_x\text{-}1]$. Once they are read out, $y(k)$ can be obtained by right shifting each $y_f(k,b_1)$ b_1 bits and summing them over b_1. Next, some different mechanisms for hardware realization [13] are presented which is a trade between arithmetic and memory.

(a). single memory

$$y(k) = \sum_{b_1=0}^{B_x-1} \sum_{i=0}^{N-1} h(i)\cdot x(k\text{-}i,\ b_1)\cdot 2^{-b_1} \tag{2.13}$$

Two-dimensional convolution with $h(\cdot)$ word sequence as one and $x(\cdot)$ bit string as the other.

(b). memory partition

$$y(k) = \sum_{b_1=0}^{B_x-1} \sum_{q=0}^{Q-1} \sum_{i=qP}^{(q+1)P-1} h(i)\cdot x(k\text{-}i,\ b_1)\cdot 2^{-b_1} \tag{2.14}$$

If 2^N is too big, add one more dimension by partition N points of $\{ h(i) \}$ into Q blocks with each block having P words.

(c). L bits per word addressing of partial products

$$y(k) = \sum_{g=0}^{G-1} \left(\sum_{l=0}^{L-1} \sum_{i=0}^{N-1} h(i)\cdot x(k\text{-}i,\ g\cdot L+l)\cdot 2^{-l} \right) \cdot 2^{-g\cdot L} \tag{2.15}$$

If B_x is too long (too many addressed-out terms), add one more dimension by partition $x(\cdot)$ word into G groups with each group having L bits such that $G \cdot L = B_x$. In this way, $N \cdot L$ bits are used to address memories at a time. This is called multiprecision arithmetic. Also note we save L times of arithmetic operations at the cost of memory size increace ($2^N \rightarrow 2^{LN}$).

(d). Input data word−partition

$$y(k) = \sum_{g=0}^{G-1} \sum_{l=0}^{L-1} \left(\sum_{i=0}^{N-1} h(i) \cdot x(k-i,\ g \cdot L+l) \right) \cdot 2^{-(g \cdot L+l)} \tag{2.16}$$

Same form as (c) but is 1−bit per data word addressing, and needs $(G \cdot L-G)$ more additions in getting memory saving as 2^N size. It is a trade-off between memory layout and bus wiring when compared with (a).

(e). Mixed of memory partition and input data word partition

$$y(k) = \sum_{g=0}^{G-1} \sum_{q=0}^{Q-1} \left(\sum_{l=0}^{L-1} \sum_{i=qP}^{(q+1)P-1} h(i) \cdot x(k-i,\ g \cdot L+l) \cdot 2^{-l} \right) \cdot 2^{-g \cdot L} \tag{2.17}$$

Mixture of (b)'s $h(k)$ word block and (c)'s $x(k)$ bit group, constitute a four-dimensional convolution.

In conclusion, it is possible to have various distributed arithmetic structures by the techniques of bit grouping, word blocking, and concurrent processing. As discussed in [11], whenever digital filter structure is described down to the bit level, the size of

grouping or blocking is not necessarily right at multiple times of the data sequence's word length. (2.12) is just one of the extreme case—the convolution of bitstring and wordstring.

CHAPTER 3

A 4–TAP CONVOLUTION PROCESSING UNIT

3.1 CHIP ARCHITECTURE

Digital signal processing involves a large amount of input data which require repetitive calculations. Three classes of computer architectures are usually applied to achieve a high computational throughput [14], including bus-based, pipeline, and parallel. The following 4-tap convolution chip is thus designed in the form of parallel and pipelined architecture [15].

- Parallel and Pipeline

To have input signal processed in real time or with a reasonable delay, the chip shown in Fig. 3-1 is designed in a combination of parallel and pipelining architecture. Parallel processor architecture requires duplication of the hardware as the price to increase throughput and reduce data latency, while pipeline processing does not require this replicating. Pipelining is best suited for special purpose applications where data flow and hardware requirements are fixed. Once data enter the pipeline, a prescribed rigid sequence of operations is performed. Control is only required to initiate the processing sequence of operations, After a certain latency period, data exit the pipeline. Pipelining can improve latency by increase bandwidth, however also result in higher gate count due

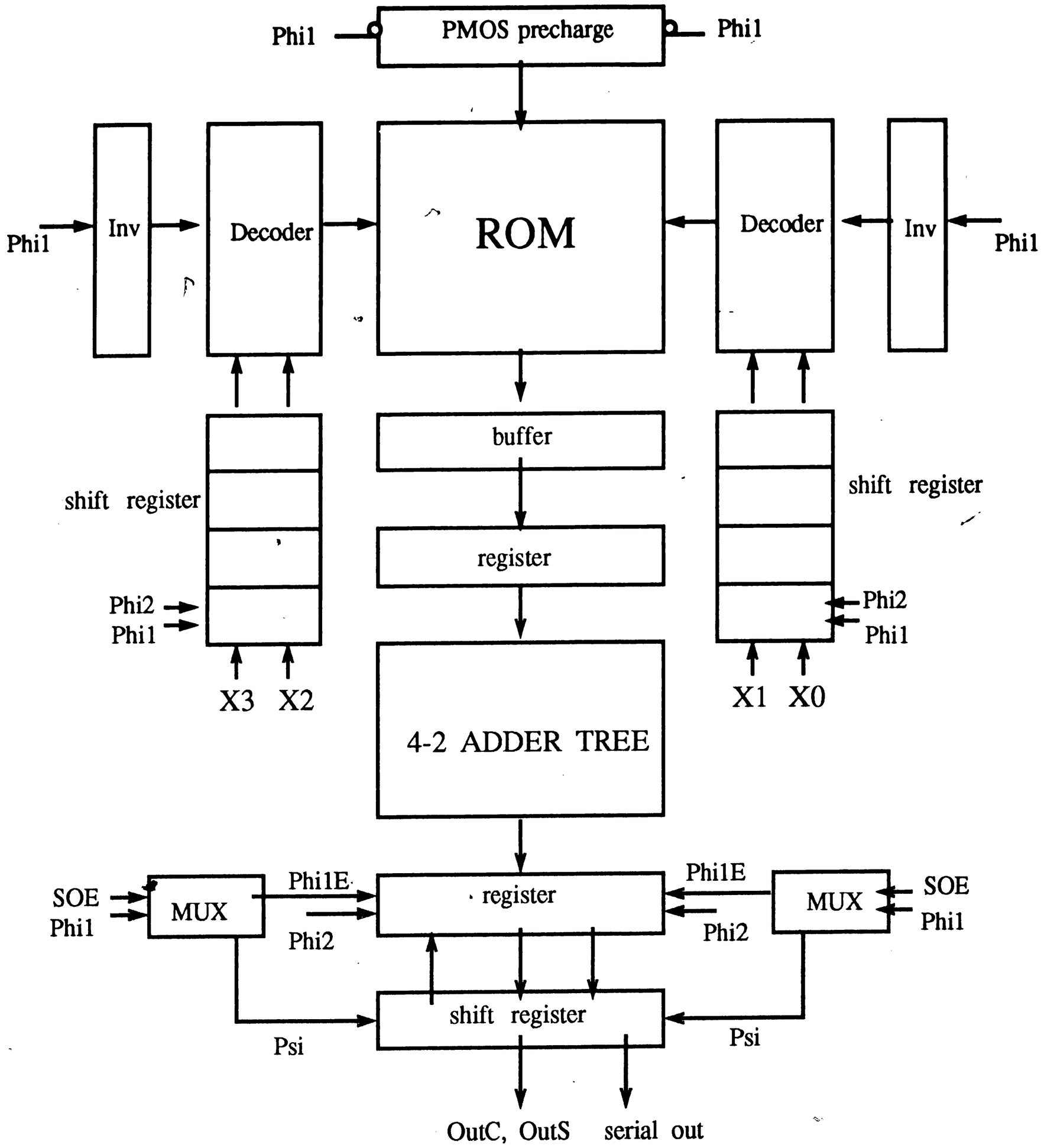


Fig. 3-1: Chip architecture of the 4-tap digital filter

to the additional latches [16].

- Clocking scheme

  A two-phase non-overlaping clock scheme is used.

3.2 VLSI IMPLEMENTATION

The basic cells and circuit elemnets used to build up a VLSI chip are crucial part in defining the speed and performance of the chip. A good example [17] is the investigation made on exclusive-OR logic gate. Even such a simple logic can vary a lot in its implementations. The consideration includes speed, number of transistors (layout size), and logic family planned to use. The same philosophy could be applied to other basic gates which are briefly reviewed as the follows:

- Register

A static cmos latch is used. Note the feedback inverter must be weak enough to remove the potential of charge sharing (signal fighting). Pass transistor replaces transmission gate saves transistor count but sacrifices signal degradation and the current leakage through the slightly turned on pmos transistor(power dissipation), Fig. 3−2.

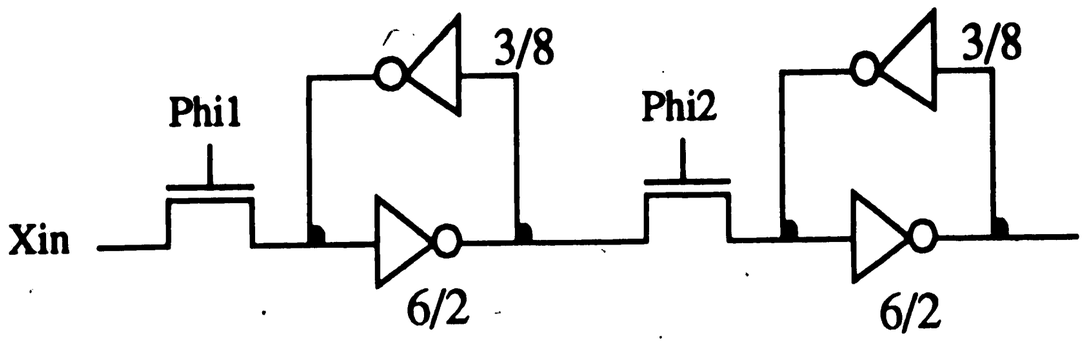
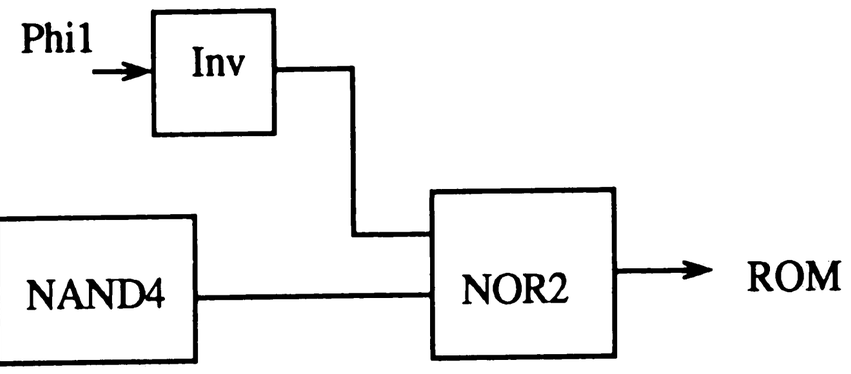
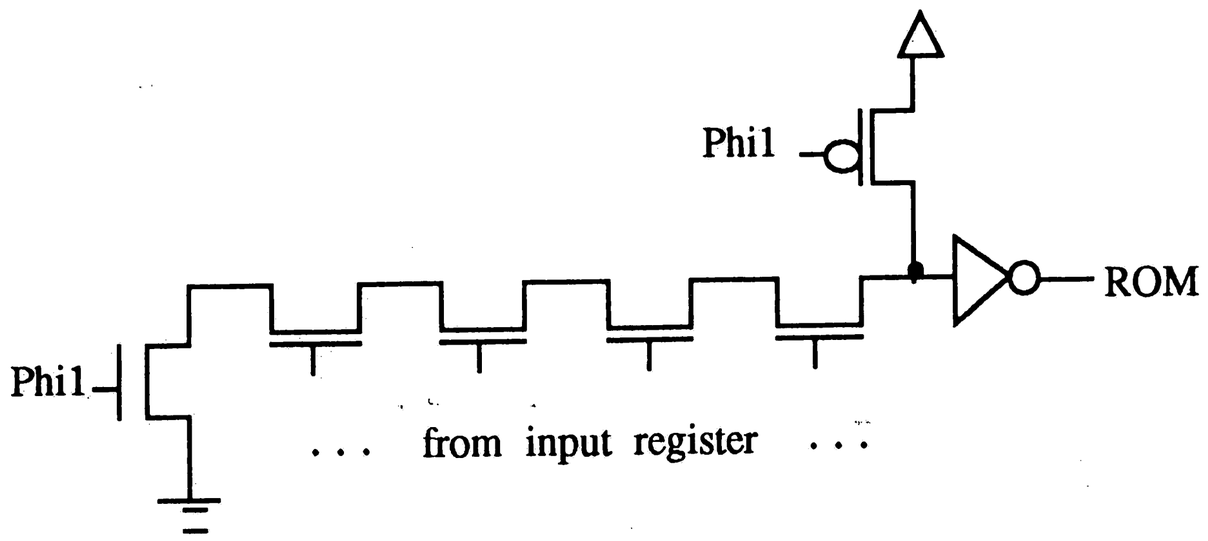


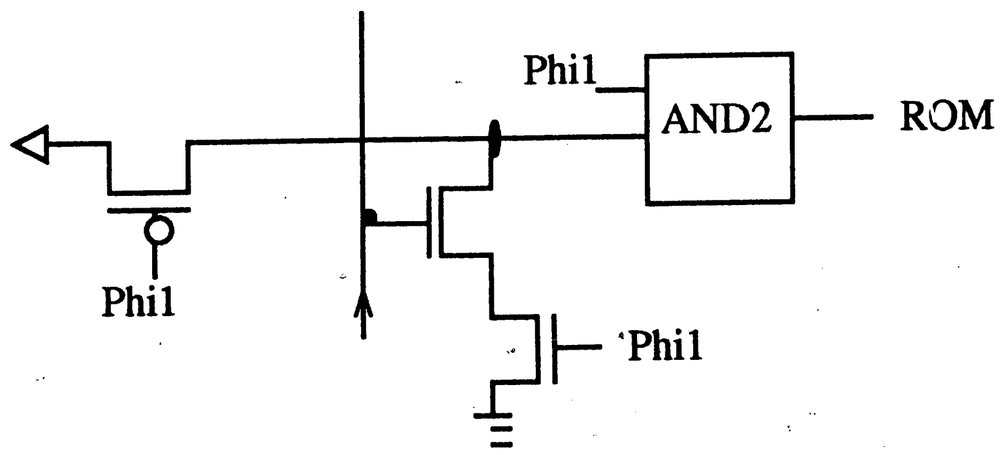
Fig. 3-2: register circuit



(A) static 4-input NAND



(B) AND plan with domino output



(C) NOR plan

Fig. 3-3: decoder circuit

• Decoder

When considering speed and testability, Fig. 3−3(a) static-nand decoder plus nor gate qualified clock is selected from the others two; (b) and (c). Owing to ROM is a precharge logic, its precedent signal−decoder output must remain logic low during the precharge period. This purpose can be obtained by using qualified clock as in (a) or domino logic output in (b), or add an evaluate nmos transistor at the bottom of ROM plane.

• ROM

The access time is a major factor in memory design. The used memory cell [18] is simulated using the method suggested by Glaeser and Dobberpuhl. The result gives 8ns access time in a size of 16 x 18 bits ROM. One other feature in this ROM is its automatic generation of m2c (metal-2 contact) photomask. A "C" program which takes digital filter's coefficients as input from designer generates a m2c layout file in MAGIC format. The architecture is very flexible and feasible for automatic design of digital filter chips.

• 4-2 Adder

It has been obvious that multiplier is not required in our chip. But adder circuit is still heavy in the structure. A special designed 4−2 adder [19] is used to construct the adder tree. It reduces the stage count in the tree from $\log_{1.5}N$ to $\log_2 N$ when compared

with using conventional full adder implementation, where N is the total number of addends.

3.3 SIGN BIT TREATMENTS

In distributed arithmetic structures, each point of the filter output is obtained by summing up B_x numbers (AA_i, i=0, 1, ..., B_x-1). These numbers are addressed out from memories by the input {x(k)}. This operation involves two problems; 1) 2's complement addition requires sign bit extension, and 2) sign bits of {x(k)} should address out a negative AA_0, recall (2.8).

For the first problem, the straight-forward method is to do sign bit extension for each AA_i to the sign bit position of AA_0. But it would cause a large fanout at the memory output and slow down the circuit. The second problem can be solved by taking one's complement of AA_0 and then plus 1. But the difficulty is to handle the extra "1" in the adder tree. H. Yamauchi et al. [20] suggested a "add one" method in their multiplier design. Their algorithm can be modified as the follows to fix the mentioned two problems.

Suppose 4 numbers addressed out, and each is 4-bit long. After weight−shift between AA_i and sign bit extension, we compute the total sum as,

$$\text{total} = AA_3 + AA_2 + AA_1 + AA_0$$

$$= \left(a_s \sum_{i=3}^{6} 2^i + \sum_{i=0}^{2} a_i \cdot 2^i\right) + \left(b_s \sum_{i=4}^{6} 2^i + \sum_{i=1}^{3} b_i \cdot 2^i\right) + \left(c_s \sum_{i=5}^{6} 2^i + \sum_{i=2}^{4} c_i \cdot 2^i\right)$$

$$+ \left(d_s \cdot 2^6 + \sum_{i=3}^{5} d_i \cdot 2^i\right). \qquad (3.1)$$

where a_s, b_s, c_s, d_s are sign bits of AA_i respectively. Then, let's consider the sign bit extension part only and use a replacement, $a_s = 1 - \bar{a}_s$. We have,

$$S = (1 - \bar{a}_s)(2^7 - 2^3) + (1 - \bar{b}_s)(2^7 - 2^4) + (1 - \bar{c}_s)(2^7 - 2^5) + (1 - \bar{d}_s)(2^7 - 2^6)$$

$$= (\bar{a}_s \cdot 2^3 + \bar{b}_s \cdot 2^4 + \bar{c}_s \cdot 2^5 + \bar{d}_s \cdot 2^6) - (2^3 + 2^4 + 2^5 + 2^6)$$

$$+ (4 - (\bar{a}_s + \bar{b}_s + \bar{c}_s + \bar{d}_s)) \cdot 2^7$$

$$= (\bar{a}_s + 1) \cdot 2^3 + \bar{b}_s \cdot 2^4 + \bar{c}_s \cdot 2^5 + \bar{d}_s \cdot 2^6 + (3 - (\bar{a}_s + \bar{b}_s + \bar{c}_s + \bar{d}_s)) \cdot 2^7 \qquad (3.2)$$

The last term aboved is at 2^7—weight and can be discarded. The remained expression is interpreted as:

a) invert AA_3 sign bit and add it to the left of the sign bit.

b) invert all other AA_i sign bits.

Next, we discuss the problem resulted from the sign bits of {x(k)}. In the aboved example, AA_0 is the number correspondent to this addressing bit, and it should be negated. Such that, we have

$$(2\text{'s of } AA_0) = (1\text{'s of } AA_0) + 1 \tag{3.3}$$

After AA_0 left-shift 3 bits, the "1" here actually means $1 \cdot 2^3$. Use the fact $2^3 = 2^2 + 2 \cdot 2^1$, this "1" can be replaced by modifying AA_0 and AA_1 as:

$$AA_0 = \left(d_s \cdot 2^6 + \sum_{i=3}^{5} d_i \cdot 2^i\right) + 2^2 + 2^1, \text{ and}$$

$$AA_1 = \left(c_s \sum_{i=5}^{6} 2^i + \sum_{i=2}^{4} c_i \cdot 2^i\right) + 2^1 . \tag{3.4}$$

In next section, it will be explained that sign bit treatments actually can be done in ROM part. Instead of using (3.4), we can just take 2's complement of AA_0 before writing it into the memory. In summary, the difficulties with the sign bit can be fixed in a hardware-economic way by the algorithm belowed,

Table–I: Sign Bit Treatments

Assume totally N addends AA_0, AA_1, . . . , AA_{n-1} are to be summed up, and AA_0 is the highest-weight number.

a) invert the sign bit of AA_{n-1} and add it to the left of the sign bit.

b) invert sign bit in each AA_1 --- AA_{n-2}.

c) invert every bits in AA_0 except the sign bit.

d) do the weight-shifting of AA_i for addition operation.

e) in AA_0, the next two bits to the right of the original LSB are set as "1".

f) in AA_1, the bit to the right of the original LSB is set as "1".

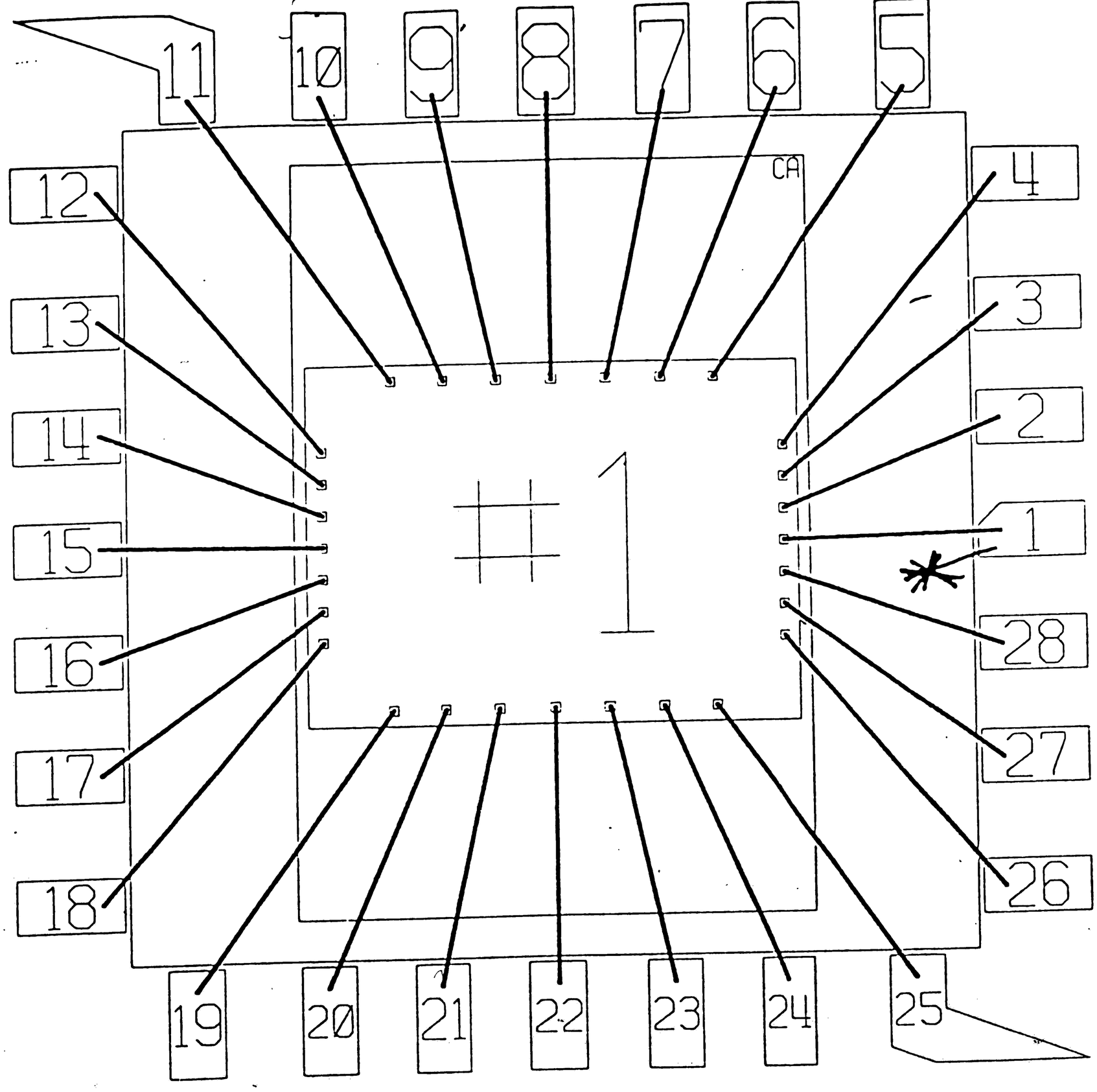
g) sum them up.

3.4 LAYOUT CONSIDERATIONS

A special feature with the chip layout is to complete the sign bit treatments in ROM part. This would save a lot of river routings from ROM's outputs to the 4−2 adder tree but with a little increasing of word-length from 18-bits to 21-bits in ROM. In other words, partial products are pre-calculated, shifted, and sign compensated then randomly programmed into ROM according to their ordering happens in the adder tree.

Special efforts are made to keep sub-cells pitch-matched such that there is almost no channel routing between modules. This is very important not only for a layout saving in the current desgin but also a favor for future's expansions based on this 4-tap unit.

The chip is packaged in 28-pin, 4.6×3.4 mm^2 ceramic DIP. Its bonding diagram and die photo are shown in Fig. 3−4 and 3−5, respectively.



M8CHCA 1

357-NSF-ACLAS/LEHIGH-CSEE

#1: 26455\FILTER -- 28P46X34

DIP28: 12 PARTS

Fig. 3–4 Bonding Diagram

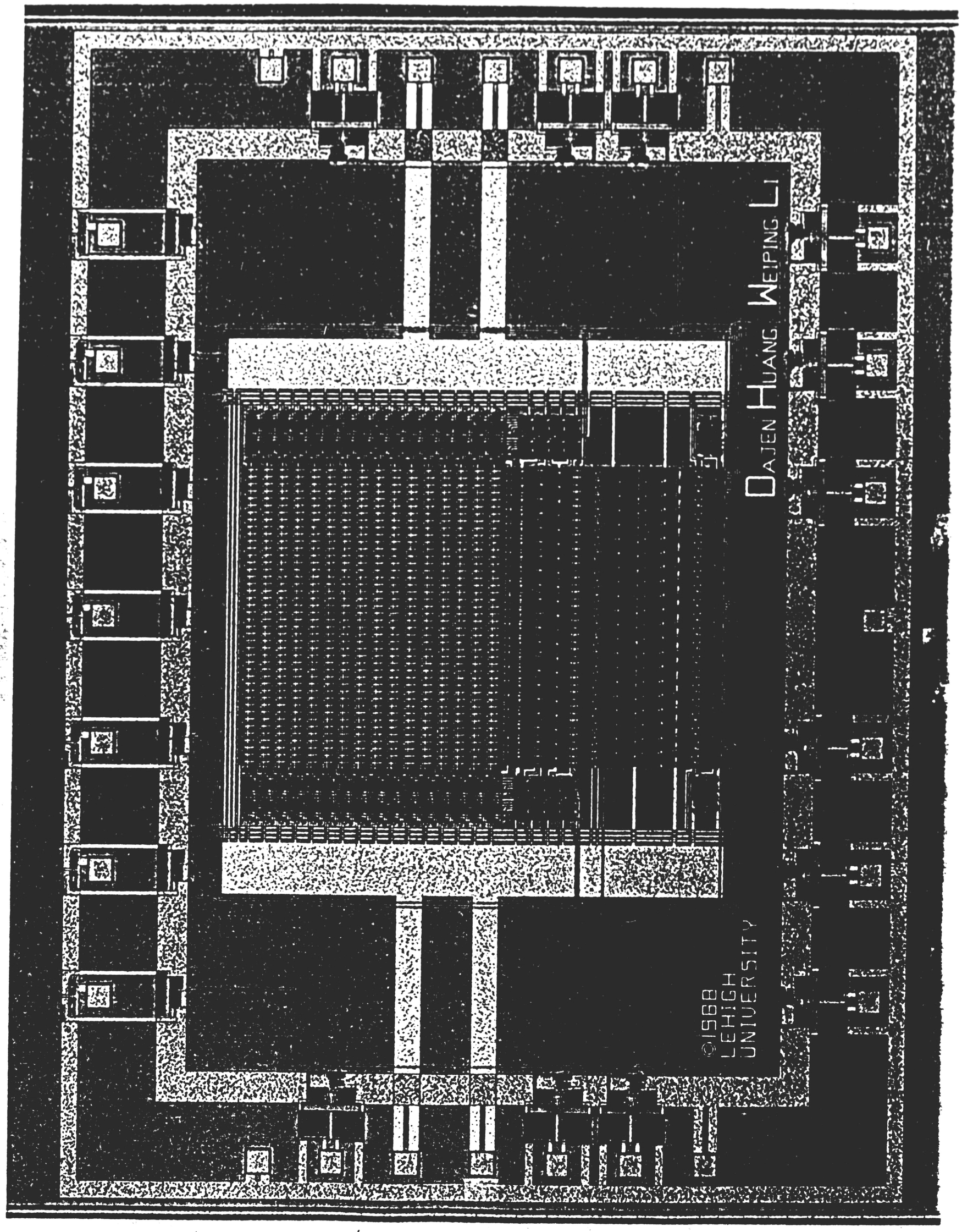


Fig. 3—5 Bar Photo

CHAPTER 4

CHIP DESIGN METHODOLOGY

4.1 DESIGN PROCEDURE

The complexity of VLSI design makes it impractical for human being to carry through the design without the help of computer. A well-followed design procedure coupled with some software tools is the only way to complete a VLSI design.

In Fig. 4−1, VLSI implementation starts from algorithm development which is verified by software simulation. From then it switches to the real chip design. After worked out the chip architecture, behavior modeling is one step further in verifying the feasibility of the developed algorithm. A high level programming language−C is used in this step. BSIM is an interface C program to help designer in simulation. Next is NET for transistor network description, and RSIM for logic simulation. Certainly, all circuit elements must be decided and critical path is checked by detailed circuit simulator SPICE in the same time. After this, chip layout is followed. MAGIC is used for cell layout, and other few programs including "cellframe" to extend cell terminals, "pwrframe" to route power and ground lines, "pad2frame" and "framegen" to generate a pad frame, and chipframe to merge all aboved. Later, Magic router is used to complete routing between the datapath and pads. Finally, layout is extracted by EXT2SIM and

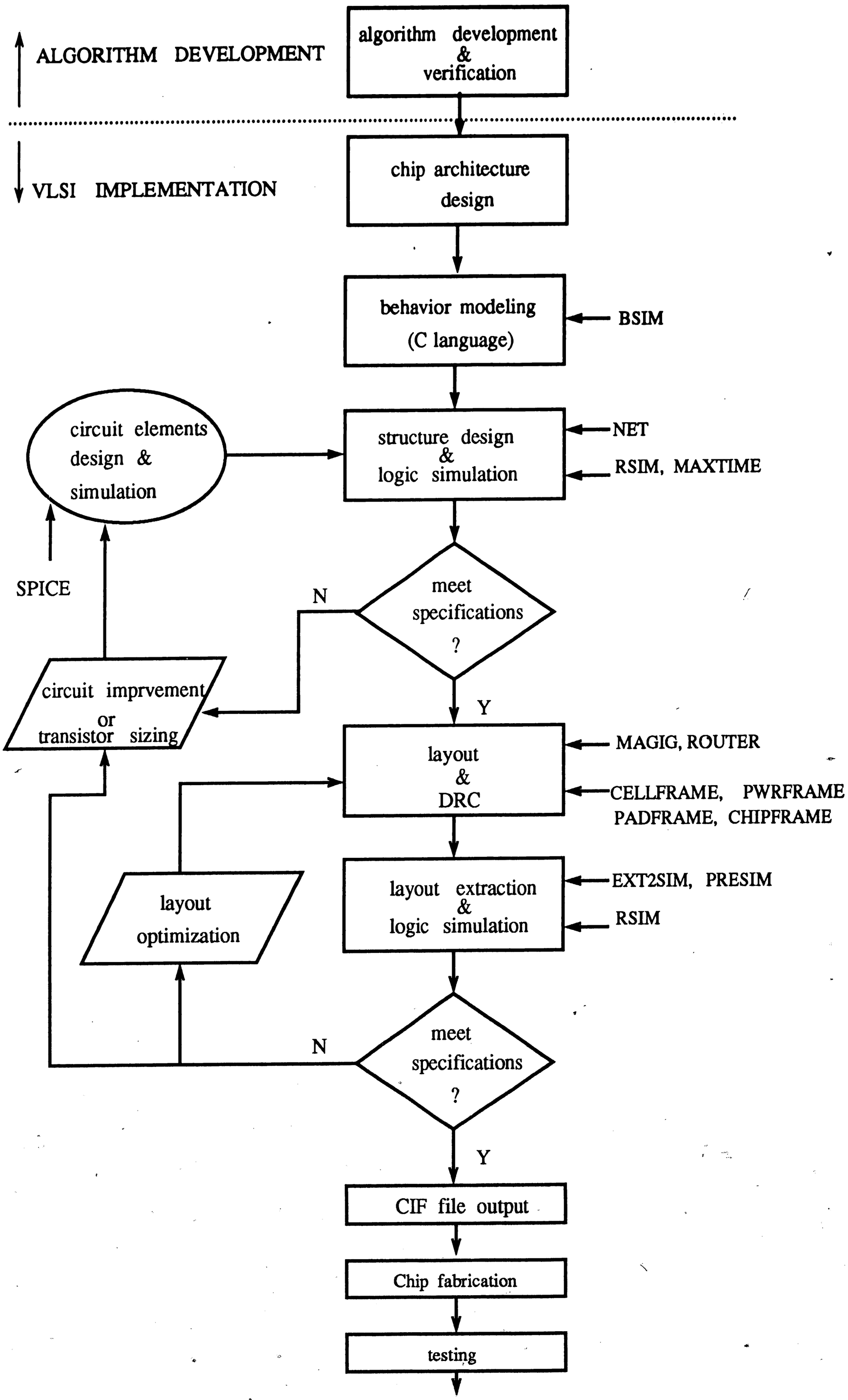


Fig. 4-1: VLSI Design Flowchart

simulated using RSIM.

In behavior modeling, structure simulation, and extracted layout simulation, an identical set of test vectors should be applied all the time such that cross checking could be carried through the design. If the final chip's performance doesn't meet the specification, further circuit change or layout optimization must be done and would induce different depth of re-work.

4.2 TESTABILITY

The key to design a "testable" VLSI chip relates to two concepts called controllability and observability. Some special techniques stem from this need include LSSD (Level Sensitive Scan Design) [21], BILBO [22], Design for Autonomous Test [23], and so on. But, it would be trading some area (increased gate count) and possibly some speed (extra loading) to achieve this level of testability.

In our design, speed testing is a problem. Conservatively speaking, a 20 MHz. high frequency tester is required if we assume the chip's max. delay is 50ns. One way to solve this problem for a combinational circuit with or without pipeline registers is to hold both Phi1 and Phi2 high, (Fig. 3−1) such that a toggled input will pass through the data path and showing change on the output. The total delay between input and output waveforms divided by the number of pipeline stages is an estimation of chip's

throughput. However, this possibility is excluded in this project due to the use of precharge logic in ROM circuit. Another trick is therefore considered and shown in Fig. 4−2. In this way, it is possible to test output's T_{plh} using general waveform generator and oscilloscope. T_{phl} can't be tested by this special method because ROM's output can't go from discharge to charge during Phi1 hold high.

The second testing problem is the pin-count limitation. This makes some of the output signals unreachable from pins. In the chip, we need 21-pins for carry-bit outputs and 21-pins for sum-bit outputs. The package used is 28-pins. An approach similar to LSSD structure is used to solve this problem. It is indicated in Fig. 4−3, a parallel/ serial mixed shift register.

4.3 CHARACTERIZATION OF THE DIGITAL FILTER

A. Functional Tests

Depending on the value of the SOE pin, the chip can operate in normally parallel output mode or in serial output mode. A complete functional testing should include connectivity tests and matrix type test patterns. Under this detailed tests, wafer processing defects could be identified if any.

As to equipments used, the BSIM program also has the capability to generate

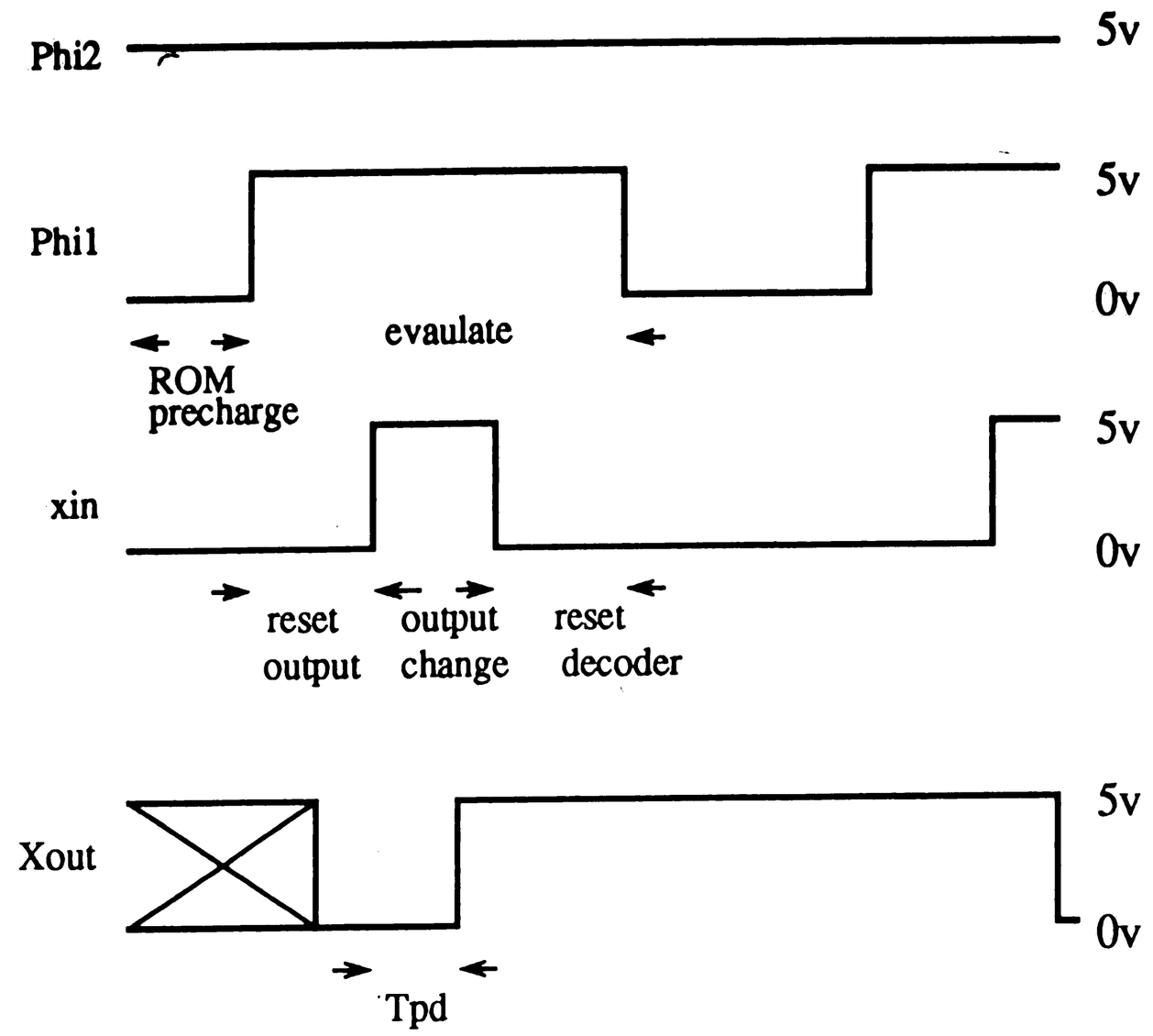


Fig. 4-2: Test pattern for performance testing

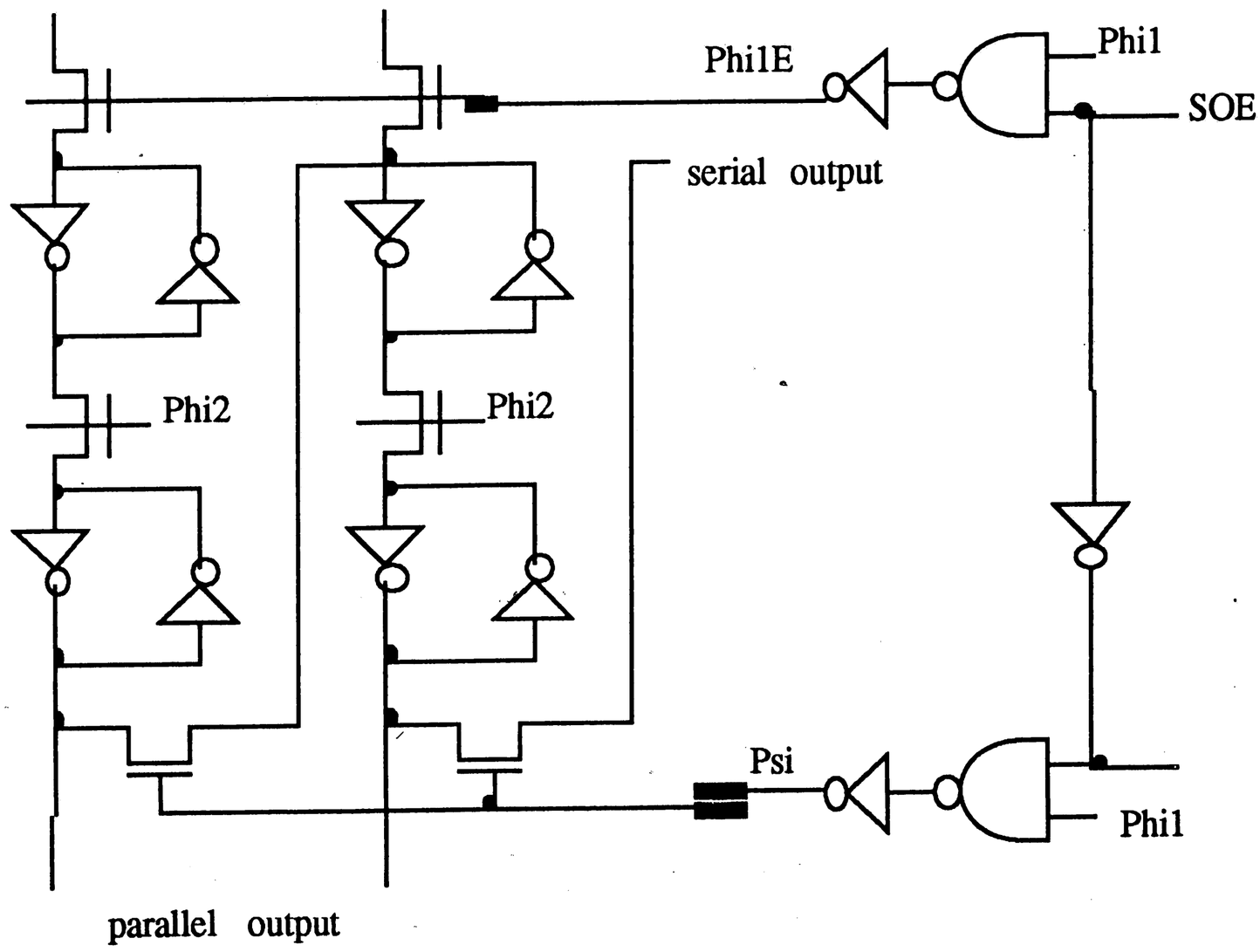


Fig. 4-3: Parallel/serial shift register

test vectors in Sieve format [24] and GenRad format [25] for functionality and performance testing. "Sunkits" is the simple tester used here.

A behavior model for the chip and its BSIM, RSIM input/output files are listed in appendix. The completed testing gives 75% yield (9 out of 12)[1,2].

B. Performance Tests

Test pattern is the one suggested in section two. Again, a behavior model with command input file were prepared. The test result gives 82ns Tp_{lh}[3].

---

1. Special thanks to James B. Burr at Standford University who did the chip testing.

2. The chip was fabricated through the MOSIS service.

3. Data is obtained from RSIM simulation. The chip's actual speed is not available at this moment.

CHAPTER 5

A DISCRETE COSINE TRANSFORM CHIP

5.1 INTRODUCTION OF THE DCT

Discrete cosine transform(DCT) is one of the most popular techniques used in video data compression. The DCT, developed in 1973 [26], has a performance verified as the most closest one in the class of orthogonal transformations to that of the Karhunen-Loeve transform (KLT) which is known to be optimal in some sense but has no general algorithm to do fast computation. To get a feeling why image data compression is needed, consider that a full-motion color video signal requires nearly 100Mbits/sec! Even a still picture at 512×512 resolution requires 6.3 Mbits (3×8 bits/pixel ,with RGB three colors) of storage. Now, if an image is broken into 32×32 blocks of data and transformed by DCT at a compression rate 32:1 (.75 bits/pixel). That image will take only approximately 1.9ms to transmit its 6.3M/32 bits data (use the 10μs data in Table−II). This is a very excellent result.

The DCT transformation consists of a set of basis vectors that are sampled cosine functions.

$$X(0) = \frac{\sqrt{2}}{N} \cdot \sum_{n=0}^{N-1} x(n),$$

$$X(k) = \frac{2}{N} \cdot \sum_{n=0}^{N-1} x(n) \cdot \cos\frac{(2n+1)k\pi}{2N}, \quad k=1, 2, ..., N-1. \tag{5.1}$$

where x(n) is input data sequence, X(k) is the kth DCT component, and $\{1/\sqrt{2}, \cos((2n+1)k\pi/2N)\}$ is the set of basis vectors which is also known as a class of discrete Chebyshev polynomials. Traditionally, the DCT can be computed using the fast Fourier transform (FFT) [27] [28].

$$X(0) = \frac{\sqrt{2}}{N} \cdot \sum_{n=0}^{N-1} x(n)$$

$$X(k) = \frac{2}{N} \cdot \mathrm{Re}\left\{ e^{-j\frac{k\pi}{2N}} \sum_{n=0}^{2N-1} x(n) \cdot e^{-j\frac{2\pi}{2N}kn} \right\}, \quad k=1, 2, \ldots, N-1 \qquad (5.2)$$

So the DCT realization can be accomplished by using multipliers to implement the butterfly structure of various FFT algorithms. The fast algorithm [29] proposed by Chen, through a direct decomposition of DCT matrix, provided another way of computing the DCT. Nevertheless, all these approaches focused on reducing the number of multipliers. These approaches are good for computing the DCT using general purpose computer but not so efficient for VLSI implementations.

Distributed Arithmetic is again feasible in DCT realization by noting the following facts: 1) The transform coefficients, $\{1/\sqrt{2}, \cos((2n+1)k\pi/2N)\}$, are fixed constants for a certain size DCT. 2) The matrix-vector product of transform operations can be realized by a large number of concurrent vector inner products. This new

approach can end up with a new architecture of DCT which consists of registers, memories, and adders only; no multiplier or butterfly structure is needed.

5.2 THE NEW COMPUTING ALGORITHM

VLSI technology today provides the possibilities of the DCT hardware implementation using distributed arithmetic algorithm. Some real examples have been reported in recent years [30] [31] [32] [33] [34], and [34] even presented a completed chip.

The two dimensional patterns in image coding, corresponding to the rows and columns of an data matrix, require a two-dimensional DCT. The two-dimensional DCT of size N×N can be defined as:

$$Z = C^t \cdot X \cdot C \tag{5.3}$$

with C is the Nth-order DCT matrix, C^t is the transpose of C, and X is the input data matrix. By the row-column decomposition technique shown in Fig. 5-1, T. C. Chen et al. used bit-serial and bit-parallel for data I/O, decimation-in-frequency and ROM-partition in reducing memory size. They constructed a 16x16 DCT chip with 14.3 MHz sample rate of video data for real-time processing. This is a straight-forward application of distributed arithmetic to inner product computation between two matrices with little special investigation on the nature of DCT mathematical structure.

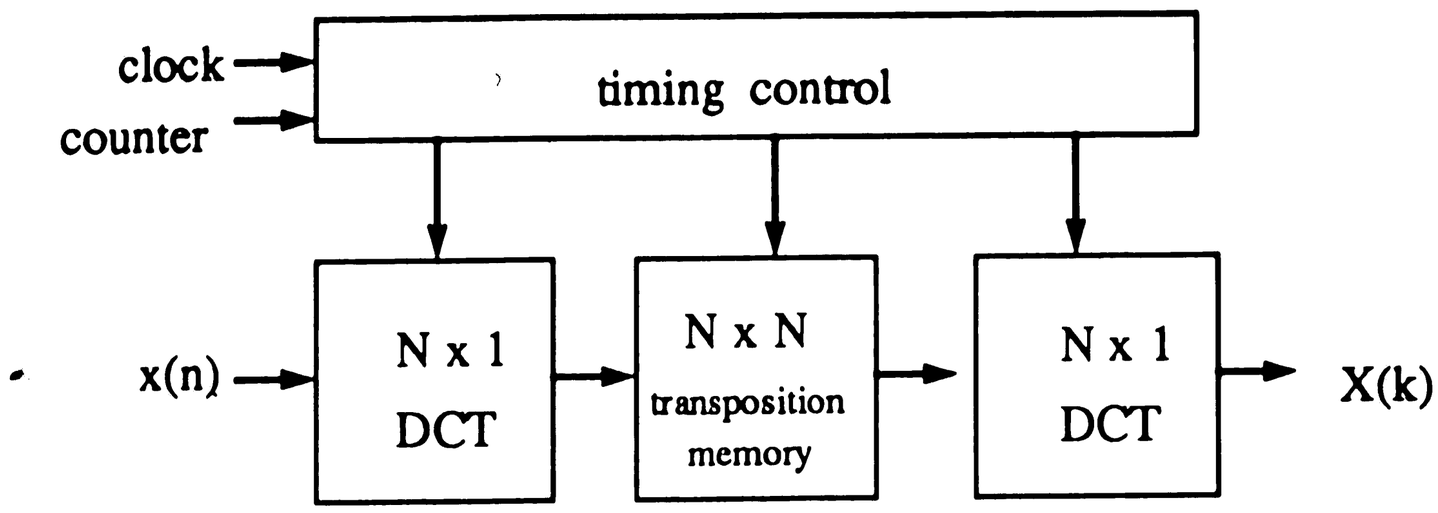


Fig. 5-1: row-column decomposition of two-dimensional DCT

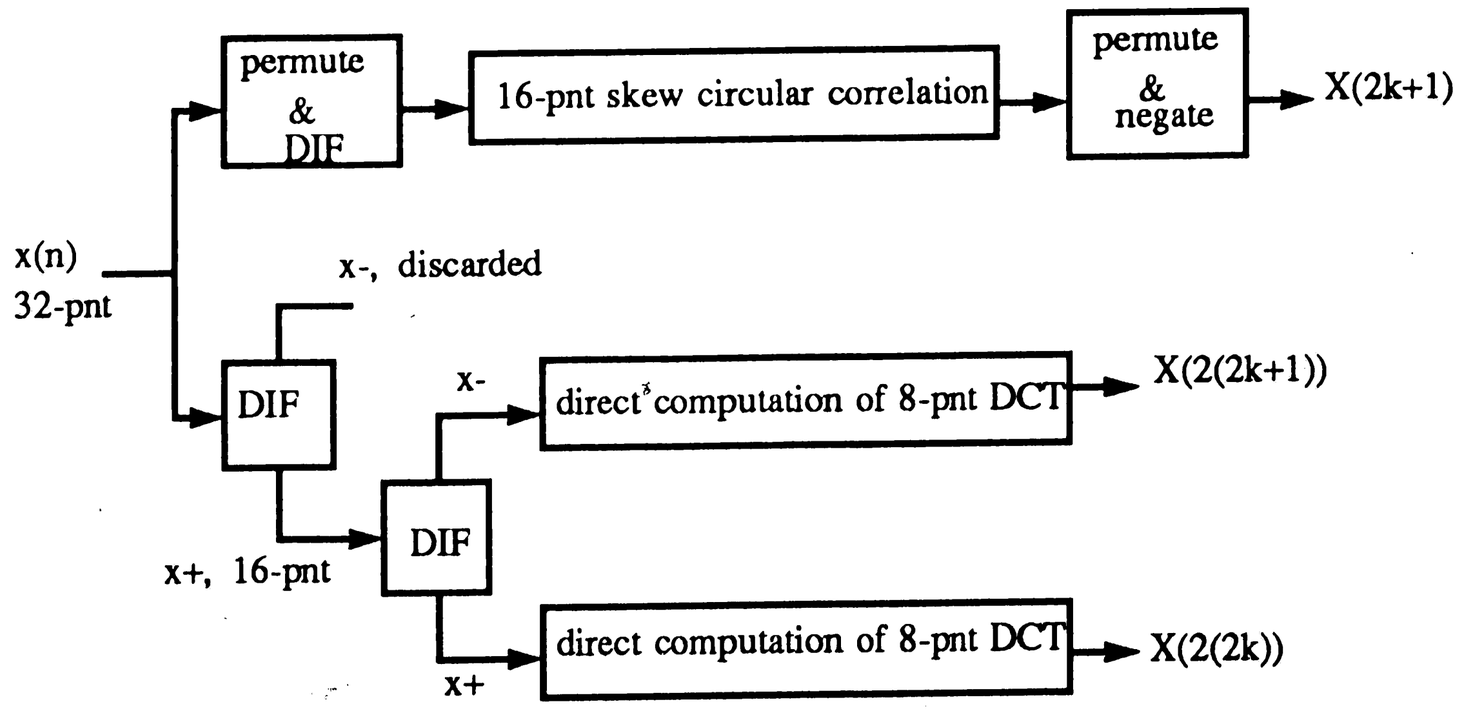


Fig. 5-2: Partition algorithm for 32-point DCT implementation

In [28], it has been pointed out that DCT computation can be regarded as polynomial products after the data matrix is permuted in some way, and these polynomial products are evaluated by distributed arithmetic. Below, a new algorithm is presented [35] which uses a different permutation and transfers DCT operation as a circular correlation between the data sequence and the transform coefficients.

- The New Computing Algorithm

The idea behind Li's algorithm is based on a one to one mapping between $\{$odd integers in [0, 4N], N=0,1,..,N-1$\}$ and $\{(-1)^l 3^i \bmod(4N)$, l=0,1 and i=0,1,...,N-1$\}$. The product relationship of input index and output index is changed to be sum relationship via this exponential operation, so a transform operation becomes a correlation operation. In short, the odd terms in DCT computation,

$$X(2k+1) = \sum_{n=0}^{N-1} x(n)\cdot\cos\frac{(2n+1)(2k+1)\pi}{2N}, \quad k=0, 1, 2, \ldots, \frac{N}{2}-1 \tag{5.4}$$

can become as circular correlation in (5.5) via (5.6), (5.7) permutations.

$$\Rightarrow \boxed{X''(j) = \sum_{i=0}^{N-1} x''(i)\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j} \bmod(4N)\right)} \tag{5.5}$$

$$x''(i) = x(n), \qquad i = 0, 1, \ldots, N-1 \tag{5.6}$$

$$n = \tfrac{1}{2}\cdot(3^i \bmod(4N)-1), \text{ if } \tfrac{1}{2}\cdot(3^i \bmod(4N)-1) < N, \text{ or}$$

$$n = (2N-1) - \tfrac{1}{2}\cdot(3^i \bmod(4N)-1), \text{ if } \tfrac{1}{2}\cdot(3^i \bmod(4N)-1) > N.$$

$$X(2k+1) = X''(j), \quad \text{for } j = 0, 1, \ldots, N\text{-}1 \tag{5.7}$$

$$k = \tfrac{1}{2}(3^j \bmod(4N) - 1), \text{ if } \tfrac{1}{2}(3^j \bmod(4N) - 1) < \tfrac{N}{2}, \text{ or}$$

$$k = 2N-1 - \tfrac{1}{2}(3^j \bmod(4N) - 1), \text{ if } \tfrac{1}{2}(3^j \bmod(4N) - 1) \geq \tfrac{3}{2}N$$

As to the even indexed terms, they are computed as a $\frac{N}{2}$ point DCT in the same way after doing recursive decimation-in-frequency on the data sequence, as described in [35]. Thus, It is shown that DCT can be computed as circular correlation by performing permutations on both input and output sequences, respectively.

Note that the direct inner product computation of DCT suggested by T. C. Chen using distributed arithmetic requires N different memory cells to store N different sets of coefficient summations, totally $N\cdot 2^N$ words are needed. Through the help of DIF and memory partition, $N\cdot\frac{N}{8}\cdot 2^4$ is the final memory size for a Nx1 DCT computing block. The new algorithm of computing the DCT as circular correlation requires B_x identical sets of coefficients, consequently, needs approximately $2\cdot B_x\cdot\frac{N}{8}\cdot 2^4$ memories. This allows a hardware realization using only half size of memory, and provides a faster operation in N=32 and B_x=8 case. More explanations are followed.

First, to recognize that distributed arithmetic is suitable in implementation of (5.5), let's review the expression,

$$X''(j) = \sum_{i=0}^{N-1} x''(i)\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j} \bmod (4N)\right) \tag{5.8}$$

$$\Rightarrow \quad X(j) = \sum_{i=0}^{N-1} x(i)\cdot C_o(i+j), \quad \text{where } C_o \text{ is periodic sequence in N}$$

$$\Rightarrow \quad X(j) = \sum_{i=0}^{N-1} x_o(i-j)\cdot C(i), \quad \text{where } x_o \text{ is periodic sequence in N}$$

$$= \sum_{m=0}^{B_x-1}\left(\sum_{i=0}^{N-1} x((i-j) \bmod (N), m)\cdot C(i)\right)\cdot 2^{-m} \tag{5.9}$$

By circulating the input data sequence, each output is obtained by solving the inner product of two matrices, and distributed arithmetic is an efficient way to do it. Before going to the chip architecture design, a detailed investigation of arithmetic properties of (5.5) is necessary for such a huge number cruncher to be integrated on a chip.

1. DIF—before permutation of the input data sequence

The basis vectors $\cos(\cdot)$ is a symmetric function in 2π range. This property can be used to decompose the primitive DCT expression as:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \Big\{x(n) - x(N\text{-}1\text{-}n)\Big\} \cdot \cos\frac{(2n+1)\cdot k\pi}{2N}, \text{ if k is odd, and}$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \Big\{x(n) + x(N\text{-}1\text{-}n)\Big\} \cdot \cos\frac{(2n+1)\cdot k\pi}{2N}, \text{ if k is even.} \qquad (5.10)$$

A N-point DCT is now traded as two $\frac{N}{2}$-point DCT. It is a big saving in computation. The consecutive decimation of the even part is permissible but at the expense of increasing irregularity in hardware [33].

2. *DIF—after permutation of the input data sequence*

Regard to the new algorithm, no wonder this cosine symmetry is preserved but in somehow different aspect due to the permutation executed on data sequence. From (5.5)

$$X(j) = \sum_{i=0}^{N-1} x(i)\cos\Big(\frac{2\pi}{4N}\cdot 3^{i+j} \bmod(4N)\Big)$$

$$= \sum_{i=0}^{\frac{N}{2}-1} x(i)\cos\Big(\frac{2\pi}{4N}\cdot 3^{i+j} \bmod(4N)\Big) + \sum_{i=0}^{\frac{N}{2}-1} x(i+\tfrac{N}{2})\cos\Big(\frac{2\pi}{4N}\cdot(3^{\frac{N}{2}}\cdot 3^{i+j}) \bmod(4N)\Big)$$

Use the fact [36], $3^{\frac{N}{2}} \bmod(4N) = 2N + 1$, $N=2^{t+1}$, $t=2, 3, 4, \ldots$ (5.11)

we get,

$$X(j) = \sum_{i=0}^{\frac{N}{2}-1} x(i)\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j}\bmod(4N)\right) + \sum_{i=0}^{\frac{N}{2}-1} x(i+\tfrac{N}{2})\cos\left(\frac{2\pi}{4N}\cdot(2N+1)\cdot 3^{i+j}\bmod(4N)\right)$$

$$= \sum_{i=0}^{\frac{N}{2}-1} \left(x(i) - x(i+\tfrac{N}{2})\right)\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j}\bmod(4N)\right),\ j = 0, 1, \ldots, N\text{-}1. \quad (5.12)$$

It proves that the N-point circular correlation is decimated as $\frac{N}{2}$-point skew circular correlation. Skew circular comes from the observation:

$$\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j}\bmod(4N)\right) = \cos\left(\frac{2\pi}{4N}\cdot(3^{\frac{N}{2}}\cdot 3^{i+j-\frac{N}{2}})\bmod(4N)\right),\ \text{when } i+j > \tfrac{N}{2}$$

$$= -\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j-\frac{N}{2}}\bmod(4N)\right). \quad (5.13)$$

Using distributed arithmetic, the interchangeable property of correlation allow us to do the skew circulation in x(n) and fix one set of $\frac{N}{2}$-point coefficients in memories.

Though (5.12) has saved us half of computations for each X(j), recall (5.7) that N-point X(j) is mapped to $\frac{N}{2}$-point odd terms of X(k) of final DCT output. This redundancy is eliminated by noting,

$$X(\tfrac{N}{2}+j) = \sum_{i=0}^{\frac{N}{2}-1} \left(x(i) - x(i+\tfrac{N}{2})\right)\cos\left(\frac{2\pi}{4N}\cdot(3^{\frac{N}{2}}\cdot 3^{i+j}\bmod(4N)\right)$$

$$= -\sum_{i=0}^{\frac{N}{2}-1} \left(x(i) - x(i+\tfrac{N}{2})\right)\cos\left(\frac{2\pi}{4N}\cdot 3^{i+j}\bmod(4N)\right)$$

$$= - \mathbf{X}(j), \qquad j = 0, 1, \ldots, \frac{N}{2}. \tag{5.14}$$

This implies that the second half $\frac{N}{2}$ points of $\mathbf{X}(\cdot)$ output can be obtained from the first half $\frac{N}{2}$ points by taking a minus sign. Finally, we have $\frac{N}{2}$ point DCT output by doing $\frac{N}{2}$ computations of $\frac{N}{2}$ point skew circular correlation.

3. *DIF—in memory part*

A third place where the cosine symmetry property can be applied is in the memory in which we store all possible summations of coefficients. A memory unit of 2^4 words (the commonly used size after partition) could be simplified as only 4 words if the coefficients are specially grouped. For example, the 16 combinations of 4 coefficients {a, b, -a, -b} is concentrated as {a, b, a+b, a-b} 4 words plus a minus sign. This allows a decreasing of memory size from 2^4 to 4. But the drawback is the increased complexity on decoder design because each four matched signals must be on-purpose collected in a group, such that they can fetch the correspondent squeezed memory, furthermore the minus sign also needs special care in hardware. The worst is all structures before the memory part will be double-sized since DIF operation is postponed.

In conclusion, the symmetric property of cosine function allows us to do DIF during the computation and it saves a lot in the processing. A choice exists in

considering "where" or say "when" is the best spot to "use out" this unique advantage. Further split on the basis vector gives cosine and sine relationship in between, obviously it doesn't do any help.

5.3 CHIP ARCHITECTURE

The 32x1 DCT module proposed in Fig. 5−3 is a mixture of the new algorithm and Chen's approach. The reason for that is the new algorithm only exhibits its superiority when N > Bx. With a usual 8-bit representation for image data, Chen's is preferred when N is not much larger than 8 because additional hardware for I/O permutation must be paid in doing the new algorithm. More detailed comparisons are summarized in the next section. Fig. 5−2 demonstrates how partition could be done to get advantages from two different algorithms. The same concept can be applied to N=64, 128, ... larger points of DCT.

Below, some features in hardware implementation are discussed:

1. chip architecture

A high degree of processing concurrency is achieved through the use of three pipelined channels in parallel. A second set of clock, Phi1d and Phi2d operates at a half frequency of Phi1 and Phi2, respectively, is used to get more efficient in hardware

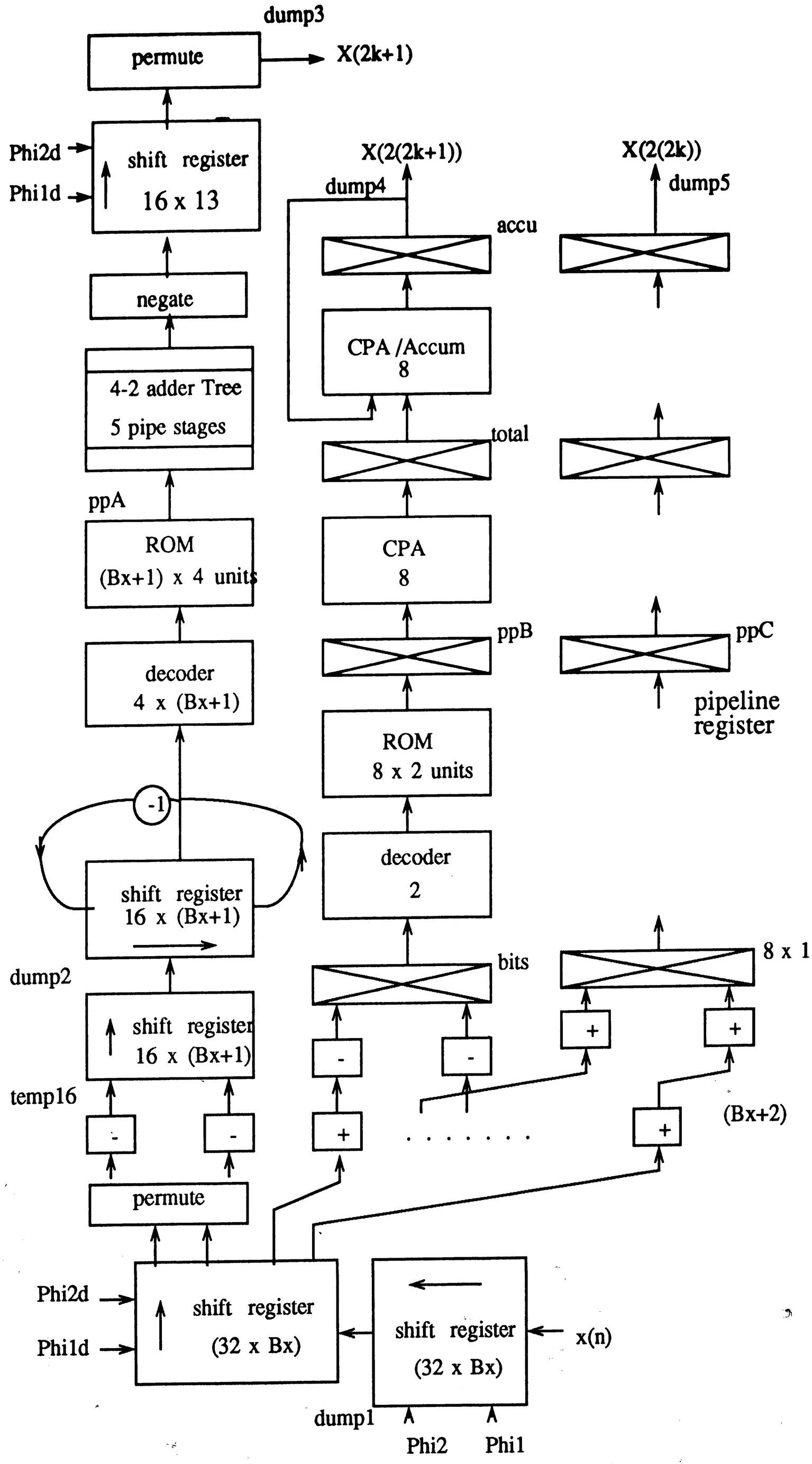


Fig. 5-3: Chip Architecture of 32x1 DCT block

utilizations.

2. permutation

Based on the algorithm, sequences permutation is required on both input and output. It would be a heavy routing if imagine a 32-words sequence permutated to be the other. One of the choices is to use (serial-in, parallel-out), (parallel-in, serial-out) structure [30]. This is another example of trading time for space.

3. control signal

Due to the use of two different algorithms in the implementation, three in-parallel pipeline channels executed in different speed. It causes the necessity of dedicated control signals to trig some particular operations in a right timing. The use of DIF is another reason to explain this increased complexity because each DIF would give one-bit increment in data sequence length. For sure, one more bit in word length takes one more cycle to complete its computation. To save the use of many counters and combinational logic, a counter coupled with a set of shift register can provide all required trig signals in circuit.

5.4 COMPARISON OF DIFFERENT COMPUTING METHODS

In this section, we compare three kinds of approaches in DCT computations: 1) using AT&T DSP16A general purpose DSP chip [37], 2) VLSI implementation using Chen's algorithm (direct inner product computation), and 3) VLSI implementation using the new algorithm (circular correlation). A 32×32 2-D DCT requires $2 \cdot 2^5 \cdot 2^5 \cdot 2^5 = 2^{16}$ multiplications for each block of transformation. One block of input means $32 \times 32 = 1024$ points of coded image data. Here we assume 9-bit data input.

Table—II Comparisons of speed performance and hardware requirements of two-dimensional 32-point DCT using different computing methods (listed value is the required number of circuit elements)

mechanism	register 1-bit	decoder 4-bit	full adder 1-bit	4-2 adder 9-bit	CPA 13-bit	ROM 9-bit	expected speed
DSP16A							4.1ms
Chen's	2816	256	64	64	64	4096	72μs
New	5896	1160	96	30	66	2176	10μs

– NOTE –

1) Speed is the time consumed to complete 32×32 data points transformation.

2) DSP16A takes 1.9 instruction cycle to complete one multiply at 33ns/cycle speed.

2) Chen's design is considered for 14.3 MHz real time video signal input, equivalent to 1/70ns operating frequency. It takes 32×32 cycles to complete.

3) New method use fully parallel and pipeline structure, hopefully gives 1/10ns operating frequency (note the computing part of the chip, Fig. 5—3, is operated

at half-frequency so that 20ns is available for addition and memory fetch). It also take 32×32 cycles to complete one block transform.

4) both Chen's and new method need another 32×32 RAM (word length 13−bit) to be the transposition memory.

5) the accuracy of DCT computation is decided by the word length used in each section's register. The number of bits used here are based on M. T. Sun's study [29], and assume the second 32×1 DCT block is the same as the first one.

From the table, we notice that the new algorithm requires half of memory size of Chen's design but pays more registers and decoders. Heavy pipelining and the algorithm itself are responsible for this extra cost. However, it is still the most attractive design in N=32 case when one recognizes the layout size ratio of 1-bit register and 1-bit memory is about 1:5, and less 4−2 adders are required (1 bit 4−2 adder has 72 transistors). Besides, it gives a seven times faster execution speed than Chen's. Based on the fabricated filter chip, This proposed 32-point DCT chip would require 107mm^2 silicon area for the datapath only (based on the cell layout in the digital filter chip).

CHAPTER 6

CONCLUSION

6.1 FUTURE DEVELOPMENTS

A 4-tap convolution unit is designed, fabricated, and tested. New DCT algorithm is developed and its VLSI implementation is proposed. Future work may include:

- 32-tap digital filter:

According to the current MOSIS 2$-\mu$m process and 2.0×1.6 mm^2 area taken by the 4-tap unit, 32-tap with 8-bit data length might be the largest tap size which we can fit into the 7.9×9.2 mm^2 padframe available from MOSIS. To accomplish such a design, a good arrangement of the memory blocks must be assured to avoid routing overhead. The discussion in section 2.3 shows some examples for this kind of consideration. One other work is to design a fast CLA (Carry Lookahead Adder) which would sum-up the last two numbers from the 4-2 adder tree output. Its maximum delay should not larger than that of 4$-$2 adder, otherwise, it would become the new critical path. Conventional CLA design has the drawback of very irregular layout. Manchester carry adder [38], [16] with carry lookahead circuit is a prospective structure. Two levels of carry lookahead must be used because the length of the two numbers is over 16-bit .

• **Digital filter silicon compiler:**

As a fact, the digital filter implemented using distributed arithmetic is very regular, flexible. Only a few types of circuit elements are needed (register, decoder, ROM, 4−2 adder, and CLA). it is very practical to consider the topic− "Automatic Digital filter design using distributed arithmetic structure". IIR (Infinite Impluse Response) filter could be also covered once a feedback path (register) is provided. The complete project would involve three parts: a) a program to generate filter coefficients according to user's input specifications, b) auto−generate behavior model and net description files, and simulate them, c) generate chip layout and simulate the extracted output. The whole idea is between cell-based system and procedure design. Designer write a program and generate the final chip layout.

• 32×32 DCT chip

Chip architecture design and behavior modeling have been completed. Structure design and chip layout should be continued to get the chip completed. Such a chip is estimated to have 200,000 transistors and take over 107mm^2 die size. In simulation, at least 1024 input vectors are required and the first block of DCT output will not be ready untill about 8×1024 cycles of clcoking. All of these imply a very heavy loading to computer resources and designer's efforts.

6.2 PROSPECTIVE

Mapping DSP algorithm into VLSI chip is a very promising topic for now and the future. As semiconductor technology continue to increase the speed, the integral density and decrease the cost of processing, more and more (new or old) sophisticated algorithms could become practical. Thus the theory and chip implementation of digital signal processing is more significant as time goes on.

"The importance of digital signal processing appears to be increasing with no visible sign of saturation. Indeed, the application in this field is expanding."

BIBLIOGRAPHY

[1] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits", *Memo ERL-M520,* U. C. Berkeley, May 9, 1975.

[2] J. B. Burr, "BSIM user's guide", Stanford University, 1987

[3] C. J. Terman, "Simulation tools for digital LSI design", *Ph. D. thesis,* MIT, 1983.

[4] C. J. Terman, "User's guide to NET, PRESIM, and RNL/NL", *VLSI memo 82-112,* MIT, July 1982.

[5] J. K. Ousterhout, et al., "1985 VLSI tools: more works by the original artists", U. C. Berkeley 1985 VLSI tools distribution.

[6] R. A. Roberts and C.T. Mullis, Digital Signal Processing. Addison Wesley, pp. 72-76, 1987.

[7] L. B. Jackson, J. F. Kaiser, and H. S. Mcdonald, "An approach to the implementation of digital filters," *IEEE Trans. Audio Electroacoust. (Special Issue on Digital Filters: The Promise of LSI Applied to SIgnal Processing),* vol. AU-16, pp. 413-421, Sept. 1968.

[8] J. F. Kaiser, "Some pratical considerations in the realization of linear digital filters," *1965 Proc. 3rd Allerton Conf. on Circuit and System Theory,* pp. 621-633.

[9] A. Croisier, et al., "Digital filter for PCM encoded signals", U.S. patent 3,777,130, Dec. 4, 1973.

[10] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. A.S.S.P.* vol. ASSP-22, pp. 456-462, Dec. 1974.

[11] C. S. Burrus, "Digital filter structures described by distributed arithmetic," *IEEE Trans. on Circuit and Systems,* vol. CAS-24, no. 12, Dec. 1977.

[12] R. C. Agarwal and C. S. Burrus, "Fast one dimensional digital convolution by multidimensional techniques," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-22, pp. 1-10, Feb. 1974.

[13] F. J. Taylor, Digital Filter Design Handbook. marcel Dekker, INC., 1983, pp. 678-700.

[14] B. C. Mckinney and F. E. Guibaly, "A multiple-access pipeline architecture for digital signal processing," *IEEE Trans. on Computers*, vol. 37, no. 3, pp. 283-290, Mar. 1988.

[15] W. Li, J. B. Burr, and A. M. Peterson, "A fully parallel VLSI implementation of distributed arithmetic," *Proceedings of IEEE ISCAS'88, Finland*, Jun. 1988, pp. 1511-1515.

[16] S. Waser and M. J. Flynn, Introduction to Arithmetic for Digital Systems Designers, CBS College Publishing, 1982.

[17] D. Huang and W. Li, "On CMOS Exclusive OR Design", *internal documentation*, Mar., 1989.

[18] L. A. Glasser and D. W. Dobberpuhl, "The design and analysis of VLSI circuits", *Addison Wesley Publishing*, 1985

[19] W. Li and J. B. Burr, "Parallel multiplier accumulator using 4-2 adders", *US patent pending*, application number 088,096, filing date Aug. 21, 1987.

[20] H. Yamauchi et al., "10ns 8x8 multiplier LSI using super self-aligned process technology", *IEEE Journal of Solid-State Circuits*, vol. sc-18, no. 2, pp. 204-210, April 1983.

[21] E. B. Eichelberger and T. W. Williams, "A logic design structure for VLSI testing", *Proc. 14th Design Automation Conference*, pp. 462-468, June 1977.

[22] B. Koenemann, J. Mucha and G. Zwiehoff, "Built-in logic block observation techniques," *Digest 1979 Test Conference*, 79CH1509-9C, pp. 37-41, October 1979.

[23] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test." *IEEE Trans. on Computer*, vol. C-30, no. 11, pp. 866-875, Nov. 1981.

[24] Sunkit−I Test Software Distribution, the MOSIS service, USC/ISI, 4676 Admiralty Way, Marina Del Ray, CA 90292−6695.

[25] "GR160/180 Programmer's guide", Part no. 2420−0136, GenRad, 510 Cottonwood Drive, Milpitas, CA 95035.

[26] N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete cosine transform," *IEEE Trans. on Computer*, vol. C-23, pp. 90-93, Jan. 1974.

[27] M. J. Narasimha and A. M. Peterson, "On the computation of the discrete cosine transform", *IEEE Trans. on Communications*, vol. COM-26, no. 6, pp. 934-936, June 1978.

[28] H. Malvar, "Fast computation of discrete cosine transform through fast Hartley transform", *Electronics Letters*, vol. 22, no. 7, pp. 352-353, Mar. 1986.

[29] W. H. Chen *et al.*, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004-1009, Sept. 1977.

[30] M. Vetterli and A. Ligtenberg, "A discrete Fourier-cosine transform chip," *IEEE J. Selected Areas in Communications*, vol. SAV-4, no. 1, pp. 49-61, Jan. 1986.

[31] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations", *Signal Processing*, vol. 6, no. 4, pp. 267-278, 1984.

[32] P. Duhamel and H. H'mida, "New 2^n DCT algorithms suitable for VLSI implementation", *Proceeding of IEEE ICASSP*, pp. 1805-1808, 1987.

[33] M. T. Sun, L. Wu, and M. L. Liou, "A concurrent architecture for VLSI implementation of discrete cosine transform", *IEEE Trans. on Circuits and Systems*, vol. CAS-34, no. 8, pp. 992-994, 1987.

[34] T.C. Chen, M. T. Sun, and A. M. Gottlieb, "VLSI implementation of a 16x16 DCT", *Proceeding of IEEE ICASSP*, pp. 1973-1976, 1988.

[35] W. Li, "A DCT algorithm," to be published.

[36] W. Li, internal documentation, Mar., 1989.

[37] D. M. Blaker, "Using the DSP16/DSP16A for image compression", *AT&T DSP Review*, vol. 2, issue 1, pp. 4-5, Winter, 1989.

[38] N. Weste and K. Eshraghian, "CMOS VLSI design", pp. 320-326, 1985.

APPENDIX 1
BEHAVIOR MODELING

This section lists behavior models of the 4−tap digital filter and the 32×1 DCT. The programs are abbreviated to keep them short. As to the simulation, only the outputs are quoted. The correspondent BSIM input command can be figured out from the outputs themselves.

- Behavior model of 4−tap Digital Filter

```
model()
{
  mix();
  S_reg();
  dec();
  ROM();
  adder(carry, sum);
  L_pipe(carry, sum, 21);
}

mix()
{
  int  i, a, b, c, d;

  for(i=0; i<N*N; i++)
  {
    a = i & 1;
    b = (i>>1) & 1;
    c = (i>>2) & 1;
    d = (i>>3) & 1;

    h_mix[i] = a*H0 + b*H1 + c*H2 + d*H3;
  }
}
```

```
S_reg()
{
 int  i, j;

 if(Xin == X)   ;
 else
   for(j=0; j<Bx; j++){
     if(Phi1==1)
       sreg[3][j].in = (Xin>>j) & 1;

       for(i=N-1; i>-1; --i){
         if(Phi1==1)
    sreg[i][j].s = sreg[i][j].in;

         if(Phi2==1)
           sreg[i][j].out = sreg[i][j].s;

         if(i==0)    ;
         else
           sreg[i-1][j].in = sreg[i][j].out;
       }
    }
}
```

```
dec()
{
 int  i, j, index;
 double  p;

 for(j=0; j<Bx; j++)
 {
  index = 0;
  if(sreg[0][j].out == X)    code[j] = select[j] = X;
```

```
    else
    {
     for(i=0, p=N-1; i<N; i++, p--)
       index += sreg[i][j].out*(int)(pow(2.0, p));
     code[j] = index;
     if(Phi1 == 1)        select[j] = code[j];
     if(Phi1 == 0)        select[j] = 0;
    }
  }
}

ROM()
{
 static int  hp[Bx], ppb[Bx];
 int  j, B, loc;

 hp[0] = ( ((h_mix[select[0]] & 0x20000) ^ 0x20000) <<1) | (h_mix[select[0]]
          & 0x03ffff);
 hp[1] = ( (h_mix[select[1]] ^ 0x20000) <<1 ) & 0x07fffe;
 hp[2] = ( ((h_mix[select[2]] ^ 0x20000) <<2 ) & 0x0ffffc) | 0x2;
 hp[3] = ( ((h_mix[select[3]] ^ 0x1ffff) <<3 ) & 0x1ffff8) | 0x6;

 for(j=0; j<Bx; j++){

 if(Phi1 == 0)   ppb[j] = 0x1fffff;

 if(Phi1==1 && select[j] != X){
  for(B=0; B<21; ++B)
    if(((ppb[j]>>B)&1 == 1) && ((hp[j]>>B)&1 == 1)){
       loc = (int) pow(2.0, (double) B);
       ppb[j] ^= loc;
    }
  pp[j].in = ppb[j] ^ 0x1fffff;
}
```

```
  }
  pipe(pp, Bx);
}

adder(outc, outs)
reg  *outs, *outc;
{
  unsigned  i, a, b, c, d, cin, cout, bcd;

  if(pp[0].out == X)  ;
else
{
  cin = 0;
  for(i=0; i<Bh+5; i++, outc++, outs++)
  {
    a = (pp[3].out >> i) & 1;
    b = (pp[2].out >> i) & 1;
    c = (pp[1].out >> i) & 1;
    d = (pp[0].out >> i) & 1;

    cout = d & (b | c) | (b & c);
    bcd = b ^ c ^ d;
    outc->in = bcd & (cin | a) | (cin & a);
    outs->in = cin ^ a ^ bcd;
    cin = cout;
   }
  }
}

L_pipe(outc, outs, n)
reg  *outs, *outc;
int  n;
{
  int  i;
```

```
if(outc->in==X) ;

else{
 if(SOE==1){                          /* normal mode */
   pipe(outc, 21);
   pipe(outs, 21);
 }

 if(SOE==0)                           /* serial out mode */
   for(i=0; i<n; i++, outc++, outs++){
    if(Phi1==1){
     outs->s = outc->out;
     outc->s = (outs+1)->out;
    }
    if(Phi2 == 1){
     outc->out = outc->s;
     outs->out = outs->s;
    }
  }
 }
}


pipe(data, n)
reg *data;
int n;
{
 int i;

 if(Phi1 == 1)
   { for(i=0; i<n; i++)   (data+i)->s = (data+i)->in;}
 if(Phi2 == 1)
   { for(i=0; i<n; i++)   (data+i)->out = (data+i)->s;}
}
```

```
Xfilter()
{
  initialize all internal nodes to be unknown states.
}
```

- Digital filter BSIM output, normal mode: parallel output of carry and sum

```
**** BSIM 1.0 ****
bsim --> w S0 C0 S1 C1 S9 C9 S10 C10 S18 C18 S19 C19
bsim --> = SOE 1
bsim --> = Xin 0
bsim --> c
S0=X  C0=X  S1=X  C1=X  S9=X  C9=X  S10=X  C10=X  S18=X  C18=X  S19=X
C19=X
bsim --> p
   sreg[0]= X X X X
   sreg[1]= X X X X
   sreg[2]= X X X X
   sreg[3]= 0 0 0 0

code(0)=X  code(1)=X  code(2)=X  code(3)=X

The output of ROM (Partial Product) are:
pp0=X  pp1=X  pp2=X  pp3=X
outc=X
outs=X
.
. # doing the same way, continually input Xin as 7, 0, 7, 1
.
bsim --> = Xin 2   # we assume   h(k) = {-2, 5, 2, 1} of {h0, h1, h2, h3}
bsim --> c         # Y(3) when Xin(k) = {7, 0, 7, 0}, of {x3, x2, x1, x0}
S0=0x0  C0=0x0  S1=0x0  C1=0x1  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
```

```
bsim --> p
    sreg[0]= 0 0 0 0
    sreg[1]= 0 1 1 1
    sreg[2]= 0 0 0 1
    sreg[3]= 0 0 1 0

code(0)=6  code(1)=5  code(2)=4  code(3)=0

The output of ROM (Partial Product) are:
pp0=0x40004  pp1=0x4000c  pp2=0x8001a  pp3=0xffffe

outs=01111111111111111100
outc=01000000000000000010

bsim --> = Xin 8
bsim --> c                    # X(k) = {1, 7, 0, 7}
S0=0x0  C0=0x0  S1=0x0  C1=0x1  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
    sreg[0]= 0 1 1 1
    sreg[1]= 0 0 0 1
    sreg[2]= 0 0 1 0
    sreg[3]= 1 0 0 0

code(0)=12  code(1)=10  code(2)=8  code(3)=1

The output of ROM (Partial Product) are:
pp0=0x40007  pp1=0x40000  pp2=0x8000a  pp3=0xffffe

outs=01111111111111110100
outc=01000000000000011010

bsim --> = Xin 10
bsim --> c                    # X(k) = {2, 1, 7, 0}
```

```
S0=0x1  C0=0x0  S1=0x1  C1=0x0  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
   sreg[0]= 0 0 0 1
   sreg[1]= 0 0 1 0
   sreg[2]= 1 0 0 0
   sreg[3]= 1 0 1 0

code(0)=8  code(1)=5  code(2)=0  code(3)=3

The output of ROM (Partial Product) are:
pp0=0x40003  pp1=0x4000c  pp2=0x80006  pp3=0x10000e

outs=01111111111111110111
outc=01000000000000001100

bsim --> c                    # X(k) = {-8, 2, 1, 7}
S0=0x1  C0=0x0  S1=0x1  C1=0x0  S9=0x0  C9=0x0  S10=0x0  C10=0x0  S18=0x0
C18=0x0  S19=0x0  C19=0x1
bsim --> p
   sreg[0]= 0 0 1 0
   sreg[1]= 1 0 0 0
   sreg[2]= 1 0 1 0
   sreg[3]= 1 0 1 0

code(0)=0  code(1)=11  code(2)=0  code(3)=7

The output of ROM (Partial Product) are:
pp0=0x40001  pp1=0x40000  pp2=0x80002  pp3=0xfffe6

outs=10000000000000001011
outc=01000000000000001100

bsim --> q
```

- **Digital filter BSIM output, speed test mode, use the special waveform in Fig. 4−2.**

```
**** BSIM 1.0 ****
bsim --> w S0 C0 S1 C1 S9 C9 S10 C10 S18 C18 S19 C19
bsim --> = SOE 1
bsim --> = Phi2 1
bsim --> l Phi1
bsim --> = Xin 0
bsim --> s
bsim --> h Phi1
bsim --> s
bsim --> s
S0=0x0  C0=0x0  S1=0x0  C1=0x1  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
   sreg[0]= 0 0 0 0
   sreg[1]= 0 0 0 0
   sreg[2]= 0 0 0 0
   sreg[3]= 0 0 0 0
```

The output of ROM (Partial Product) are:

```
pp0=0x40000  pp1=0x40000  pp2=0x80002  pp3=0xfffe

outs=01111111111111111100
outc=01000000000000000010

bsim --> = Xin 1
bsim --> s
S0=0x0  C0=0x0  S1=0x1  C1=0x0  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
   sreg[0]= 0 0 0 1
   sreg[1]= 0 0 0 1
   sreg[2]= 0 0 0 1
   sreg[3]= 0 0 0 1
```

```
The output of ROM (Partial Product) are:
pp0=0x40006  pp1=0x40000   pp2=0x80002   pp3=0xffffe

outs=01111111111111111110
outc=01000000000000000100

bsim --> = Xin 0
bsim --> s
S0=0x0  C0=0x0  S1=0x1  C1=0x0  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
    sreg[0]= 0 0 0 0
    sreg[1]= 0 0 0 0
    sreg[2]= 0 0 0 0
    sreg[3]= 0 0 0 0

The output of ROM (Partial Product) are:
pp0=0x40006  pp1=0x40000   pp2=0x80002   pp3=0xffffe

outs=01111111111111111110
outc=01000000000000000100

bsim --> l Phi1
bsim --> s
S0=0x0  C0=0x0  S1=0x1  C1=0x0  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
    sreg[0]= 0 0 0 0
    sreg[1]= 0 0 0 0
    sreg[2]= 0 0 0 0
    sreg[3]= 0 0 0 0

The output of ROM (Partial Product) are:
```

```
pp0=0x40006  pp1=0x40000   pp2=0x80002   pp3=0xfffe

outs=01111111111111111110
outc=01000000000000000100

bsim --> h Phi1
bsim --> s
S0=0x0  C0=0x0  S1=0x0  C1=0x1  S9=0x1  C9=0x0  S10=0x1  C10=0x0  S18=0x1
C18=0x0  S19=0x1  C19=0x1
bsim --> p
    sreg[0]= 0 0 0 0
    sreg[1]= 0 0 0 0
    sreg[2]= 0 0 0 0
    sreg[3]= 0 0 0 0
```

The output of ROM (Partial Product) are:

```
pp0=0x40000  pp1=0x40000   pp2=0x80002   pp3=0xfffe

outs=01111111111111111100
outc=01000000000000000010

bsim --> q
```

• **Behavior Model of 32×1 DCT**

```
model()
{
  Li_coeff(16, ROM1);
  Chen_coeff(8, 1, ROM2);
  Chen_coeff(8, 0, ROM3);
  counter();

  data_in(column);
  split_xn(column);

  inner_p(bits_n, ROM2, ppB, 8);
  inner_p(bits_p, ROM3, ppC, 8);
  correlate(Vx16, ROM1, ppA, Bx+1, 16);

  S_sum(ppB, total_n, accu_n, 1, 8);
  S_sum(ppC, total_p, accu_p, 0, 8);
  sum_up(ppA, Bx+1, 16);
}

Li_coeff(n, cell)
int  n;
double  cell[][16];
{
  use chapter 5, (5.5) to calculate 16 coefficients.
  mix(coeff, cell, 0, 16);
}

Chen_coeff(size, odd_even, cell)
int  size, odd_even;
double  cell[][16];
{
   use chapter 5, (5.1) to calculate 8 coefficients.
```

```
    mix(coeff, cell, 2*k, 8);
}

counter()
{
  reset some register's values and also generate Phi1d and Phi2d.
}

data_in(out_bit)
int  *out_bit;
{
  vsi_reg(xn, Vx32, N);
  if(count32==32){
    dump(Vx32, Lx32, N);
    count32=0;
  }
  lso_reg(Lx32, out_bit, N);
}

split_xn(data)
int  *data;
{
  int  i, i_map[N], xn_n16[16];

  i_permute(32, i_map);
  semi(data, i_map, Ci16, temp16, 32);
  lsi_reg(temp16, Lx16, 16);

  if(flag[Bx+1].out==1)
    dump(Lx16, Vx16, 16);

  DIF(data, Ci_n16, Ci_p16, xn_n16, xn_p16, flag[Bx].out, 32);
  DIF(xn_p16, Ci_n8, Ci_p8, xn_n8, xn_p8, flag[Bx+1].out, 16);
```

```
  for(i=0; i<8; ++i){
    bits_n[i].in = xn_n8[i];
    bits_p[i].in = xn_p8[i];
  }
  pipe_I(bits_n, 8);
  pipe_I(bits_p, 8);
}

correlate(data, ROM_id, ppout, wide, n)
reg  *data;
double  ROM_id[][16];
dreg  *ppout;
int  wide, n;
{
  s_cir(data, n);
  L_deco(data, L_code, wide, n);
  L_fetch(L_code, ROM_id, ppout, wide, n);
  pipe(ppout, wide*n/4);
}


inner_p(data, ROM_id, ppout, n)
reg  *data;
double  ROM_id[][16];
dreg  ppout[8*8/4];
{
  int  i;

  C_deco(data, C_code, n);
  C_fetch(C_code, ROM_id, ppout, n);
  pipe(ppout, n*n/4);
}

sum_up(terms, width, n)
```

```
dreg *terms;
int  width, n;
{
 double  sum, add_up();

 if(terms->out==X)
   sum = X;
 else
   sum = add_up(terms, width, n);

   vsi_reg_F(sum, out16, 16);

   if(flag[0].out==1 && out16->out!=X)
     dump_x(out16, Xk1, 16);
}

S_sum(terms, total, accu, e_o, n)
dreg  *terms, *total, *accu;
int  e_o, n;
{
 int  i;

  cpa1(terms, total, n);
  pipe(total, n);

  cpa2(total, accu, e_o, n);
  pipe(accu, n);

  if(flag[14].out==1)
    dump_e(accu, Xk2, e_o, n);
}

XDCT()
{
```

initialize the counter and reset all nodes to be unknown states.

}

• 32×1 DCT BSIM output

**** BSIM 1.0 ****
bsim --> = xn 1
bsim --> c
bsim --> p # first pnt in

0's count32=1, Phi2d=0

flag[0] = 0 flag[1] = 0 flag[2] = 0 flag[3] = 0
flag[4] = 0 flag[5] = 0 flag[6] = 0 flag[7] = 0
flag[8] = 0 flag[9] = 0 flag[10] = 0 flag[11] = 0
flag[12] = 0 flag[13] = 0 flag[14] = 0 flag[15] = 0

Vx32[31] = 1

bsim --> = xn 3
bsim --> c
do the aboved continueously to input xn= 32, 23, 49, 4, 43, 71, 18, 8, 10, 54, 0, 43, 89, 92, 38, 6, 9, 26, 31, 24, 12, 84, 32, 90, 30, 24, 38, 28, 84, 7.

bsim --> c
bsim --> p # Vx16 skew circular

1's count32=26, Phi2d=1

flag[0] = 0 flag[1] = 0 flag[2] = 0 flag[3] = 0
flag[4] = 0 flag[5] = 0 flag[6] = 0 flag[7] = 0
flag[8] = 0 flag[9] = 0 flag[10] = 0 flag[11] = 0
flag[12] = 1 flag[13] = 0 flag[14] = 0 flag[15] = 0

Vx32[0] = 30 Vx32[1] = 24 Vx32[2] = 38 Vx32[3] = 28
Vx32[4] = 84 Vx32[5] = 7 Vx32[6] = 0 Vx32[7] = 1

Vx32[8] = 2 Vx32[9] = 3 Vx32[10] = 4 Vx32[11] = 5
Vx32[12] = 6 Vx32[13] = 7 Vx32[14] = 8 Vx32[15] = 9
Vx32[16] = 10 Vx32[17] = 11 Vx32[18] = 12 Vx32[19] = 13
Vx32[20] = 14 Vx32[21] = 15 Vx32[22] = 16 Vx32[23] = 17
Vx32[24] = 18 Vx32[25] = 19 Vx32[26] = 20 Vx32[27] = 21
Vx32[28] = 22 Vx32[29] = 23 Vx32[30] = 24 Vx32[31] = 25

Lx32[0] = 0 Lx32[1] = 0 Lx32[2] = 0 Lx32[3] = 0
Lx32[4] = 0 Lx32[5] = 0 Lx32[6] = 0 Lx32[7] = 0
Lx32[8] = 0 Lx32[9] = 0 Lx32[10] = 0 Lx32[11] = 0
Lx32[12] = 0 Lx32[13] = 0 Lx32[14] = 0 Lx32[15] = 0
Lx32[16] = 0 Lx32[17] = 0 Lx32[18] = 0 Lx32[19] = 0
Lx32[20] = 0 Lx32[21] = 0 Lx32[22] = 0 Lx32[23] = 0
Lx32[24] = 0 Lx32[25] = 0 Lx32[26] = 0 Lx32[27] = 0
Lx32[28] = 0 Lx32[29] = 0 Lx32[30] = 0 Lx32[31] = 0

Lx16[0] = 4090 Lx16[1] = 4015 Lx16[2] = 25 Lx16[3] = 34
Lx16[4] = 66 Lx16[5] = 4049 Lx16[6] = 26 Lx16[7] = 4070
Lx16[8] = 4042 Lx16[9] = 83 Lx16[10] = 4073 Lx16[11] = 4
Lx16[12] = 39 Lx16[13] = 4 Lx16[14] = 4081 Lx16[15] = 4082

Vx16[0] = -1010 Vx16[1] = 1018 Vx16[2] = 943 Vx16[3] = 25
Vx16[4] = 34 Vx16[5] = 66 Vx16[6] = 977 Vx16[7] = 26
Vx16[8] = 998 Vx16[9] = 970 Vx16[10] = 83 Vx16[11] = 1001
Vx16[12] = 4 Vx16[13] = 39 Vx16[14] = 4 Vx16[15] = 1009

bsim --> c
bsim --> p # Chen's 1st set of DCT out

1's count32=30, Phi2d=1

Xk2[0] = 1103.000 Xk2[2] = -69.670
Xk2[4] = -31.006 Xk2[6] = -214.594

```
Xk2[ 8] = 149.856    Xk2[10] = -31.645
Xk2[12] = -65.075    Xk2[14] = -206.995
Xk2[16] =  51.619    Xk2[18] =  17.386
Xk2[20] = -116.116   Xk2[22] = -12.637
Xk2[24] = -105.698   Xk2[26] = -128.523
Xk2[28] =  77.038    Xk2[30] =  69.795

bsim --> c
bsim --> p                  # Li's 1st of DCT out, 3rd Lx32 dumped

3's count32=2, Phi2d=1

Xk1[ 1] = -134.653   Xk1[ 3] = -79.446
Xk1[ 5] =  81.584    Xk1[ 7] = -191.323
Xk1[ 9] =  69.559    Xk1[11] = -113.796
Xk1[13] = 124.525    Xk1[15] = -57.407
Xk1[17] = 221.764    Xk1[19] =  28.203
Xk1[21] = -35.964    Xk1[23] =  18.993
Xk1[25] = 126.963    Xk1[27] = 205.545
Xk1[29] =   7.822    Xk1[31] = -120.301
bsim --> q
```

APPENDIX 2
RSIM SIMULATION

- ***4—Tap Digital Filter***

In this section, the RSIM input command and correspondent output are listed. The testing includes operation mode, serial output mode, and speed test mode. However, the part for serial output mode is removed because it is actually same as operation mode but with SOE=0 and output taken from S0 pinout.

- RSIM input command of operation mode, carry and sum are parallel out.
  (vector define is re—edited to keep the text short)

```
vector                    xk                    chip1_0/chip_0/regA_l_0/con5_0/x3
chip1_0/chip_0/regA_l_0/con5_0/x2          chip1_0/chip_0/regA_r_0/con6_0/x1
chip1_0/chip_0/regA_r_0/con6_0/x0
|
vector                   sx0                 chip1_0/chip_0/regA_l_0/regcon_0{0}/sx0
chip1_0/chip_0/regA_l_0/regcon_0{1}/sx0
chip1_0/chip_0/regA_r_0/regcon_0{0}/sx0
chip1_0/chip_0/regA_r_0/regcon_0{1}/sx0
|
vector                  code0              chip1_0/chip_0/decA_r_0/dec_r_0{0,1}/select
chip1_0/chip_0/decA_r_0/dec_r_0{1,1}/select
chip1_0/chip_0/decA_r_0/dec_r_0{2,1}/select
.
.
.
chip1_0/chip_0/decA_r_0/dec_r_0{15,1}/select
|
vector                    pp1                   chip1_0/chip_0/addA_0/add_0{0}/IND
chip1_0/chip_0/addA_0/add_0{1}/IND
.
.
chip1_0/chip_0/addA_0/add_0{20}/IND
```

```
|
|
vector                  outs                  chip1_0/chip_0/outregA_0/outreg_0{0}/outs
chip1_0/chip_0/outregA_0/outreg_0{1}/outs
chip1_0/chip_0/outregA_0/outreg_0{2}/outs
.
.
chip1_0/chip_0/outregA_0/outreg_0{19}/outs
chip1_0/chip_0/outregA_0/outreg_0{20}/outs
|
vector                  outc                  chip1_0/chip_0/outregA_0/outreg_0{0}/outc
chip1_0/chip_0/outregA_0/outreg_0{1}/outc
chip1_0/chip_0/outregA_0/outreg_0{2}/outc
.
.
chip1_0/chip_0/outregA_0/outreg_0{19}/outc
chip1_0/chip_0/outregA_0/outreg_0{20}/outc
|
|
w outc
w outs
w pp3
w pp2
w pp1
w pp0
w code3 code2 code1 code0
w sx0
w sx1
w sx2
w sx3
w xk
|
|THIS IS OPERATION MODE
|
```

```
stepsize 1000
h outen_l_0_SOE outen_r_0_SOE
clock phi1_left 1 0
clock phi2_left 0 1
|
max on
period 1000
|
V chip1_0/chip_0/regA_r_0/con6_0/x0 0 1 0 1 1 0 0 0
|
V chip1_0/chip_0/regA_r_0/con6_0/x1 0 1 0 1 0 1 0 1
|
V chip1_0/chip_0/regA_l_0/con5_0/x2 0 1 0 1 0 0 0 0
|
V chip1_0/chip_0/regA_l_0/con5_0/x3 0 0 0 0 0 0 1 1
|
|
R
c
c
| the latest signals
pn 3 0
|
exit
```

- **RSIM output, operation mode, carry, sum are parallel out**

```
*** RSIM Version 6.0 ***
11990 nodes, transistors: n-channel=3534 p-channel=1956
xk=0000 sx3=0000 sx2=XXXX sx1=XXXX sx0=XXXX code0=X1X1X1X1X1X1X1X1
code1=X1X1X1X1X1X1X1X1                          code2=X1X1X1X1X1X1X1X1
code3=X1X1X1X1X1X1X1X1
pp0=XXXXXXXXXXXXXXXXXXXXXX      pp1=XXXXXXXXXXXXXXXXXXXXXX
pp2=XXXXXXXXXXXXXXXXXXXXXX
pp3=XXXXXXXXXXXXXXXXXXXXXX outs=XXXXXXXXXXXXXXXXXXXXXX
outc=XXXXXXXXXXXXXXXXXXXXXX
outc=XXXXXXXXXXXXXXXXXXXXXX
time = 200.0ns
time = 1000.0ns
.
.
xk=0010 sx3=0010 sx2=0001 sx1=0111 sx0=0000 code0=1111110111111111
code1=1111101111111111 code2=1111011111111111 code3=0111111111111111
pp0=00100000000000000100                    pp1=00100000000000001100
pp2=01000000000000011010
pp3=01111111111111111110 outs=01111111111111111100
outc=01000000000000000010
time = 1200.0ns
xk=0010 sx3=0010 sx2=0001 sx1=0111 sx0=0000 code0=1111110111111111
code1=1111101111111111 code2=1111011111111111 code3=0111111111111111
pp0=00100000000000000100                    pp1=00100000000000001100
pp2=01000000000000011010
pp3=01111111111111111110 outs=01111111111111111100
outc=01000000000000000010
time = 1200.0ns
xk=1000 sx3=1000 sx2=0010 sx1=0001 sx0=0111 code0=1111111111110111
code1=1111111111011111 code2=1111111101111111 code3=1011111111111111
pp0=00100000000000000111                    pp1=00100000000000000000
pp2=01000000000000001010
pp3=01111111111111111110 outs=01111111111111110100
```

outc=01000000000000011010

time = 1400.0ns

xk=1000 sx3=1000 sx2=0010 sx1=0001 sx0=0111 code0=1111111111110111

code1=1111111111011111 code2=1111111101111111 code3=1011111111111111

pp0=00100000000000000111 pp1=00100000000000000000

pp2=01000000000000001010

pp3=01111111111111111110 outs=01111111111111110100

outc=01000000000000011010

time = 1400.0ns

xk=1010 sx3=1010 sx2=1000 sx1=0010 sx0=0001 code0=1111111101111111

code1=1111101111111111 code2=0111111111111111 code3=1110111111111111

pp0=00100000000000000011 pp1=00100000000000001100

pp2=01000000000000000110

pp3=10000000000000001110 outs=01111111111111110111

outc=01000000000000001100

time = 1600.0ns

xk=1010 sx3=1010 sx2=1000 sx1=0010 sx0=0001 code0=1111111101111111

code1=1111101111111111 code2=0111111111111111 code3=1110111111111111

pp0=00100000000000000011 pp1=00100000000000001100

pp2=01000000000000000110

pp3=10000000000000001110 outs=01111111111111110111

outc=01000000000000001100

time = 1600.0ns

xk=1010 sx3=1010 sx2=1010 sx1=1000 sx0=0010 code0=0111111111111111

code1=1111111111101111 code2=0111111111111111 code3=1111111011111111

pp0=00100000000000000001 pp1=00100000000000000000

pp2=01000000000000000010

pp3=01111111111111100110 outs=10000000000000001011

outc=01000000000000001100

time = 1800.0ns

xk=1010 sx3=1010 sx2=1010 sx1=1010 sx0=1000 code0=0111111111111111

code1=1111111011111111 code2=0111111111111111 code3=1111111111111110

pp0=00100000000000000000 pp1=00100000000000001000

pp2=01000000000000000010

```
pp3=01111111111111010110 outs=01111111111111100101
outc=01000000000000000010
time = 2000.0ns      /* timing statistics, shows that max. delay is 12.9ns */
  120   129    8120    1129    82 0 386
  112   126    8112    1126    75 0 324
  112   126    8112    1126    75 0 320
```

- RSIM input command of speed test mode, use the special waveform in Fig. 4−2.

```
vector                          xk                  chip1_0/chip_0/regA_l_0/con5_0/x3
chip1_0/chip_0/regA_l_0/con5_0/x2          chip1_0/chip_0/regA_r_0/con6_0/x1
chip1_0/chip_0/regA_r_0/con6_0/x0
|
| ..... define all vectors like we did in operation mode. ....
|
w outc
w outs
w pp3
w pp2
w pp1
w pp0
w code3 code2 code1 code0
w sx0
w sx1
w sx2
w sx3
w xk
|
|
|THIS IS SPEED TEST MODE
|
|
|
stepsize 1000
w -pp0 -pp1 -pp2 -pp3
h phi2_left outen_l_0_SOE outen_r_0_SOE
clock phi1_left 0 1 1 1
l xk
clock chip1_0/chip_0/regA_r_0/con6_0/x0  0 0 1 0
t chip1_0/chip_0/outregA_0/outreg_0{19}/outs
t chip1_0/chip_0/outregA_0/outreg_0{19}/outc
```

```
c
c
|
|now, repeat it again, but step by step !!!
|
clock
l phi1_left chip1_0/chip_0/regA_r_0/con6_0/x0
s
h phi1_left
s
h chip1_0/chip_0/regA_r_0/con6_0/x0
s
l chip1_0/chip_0/regA_r_0/con6_0/x0
s
l phi1_left
s
| the latest signals
pn 3 0
|
exit
```

- **RSIM output of speed test mode.**

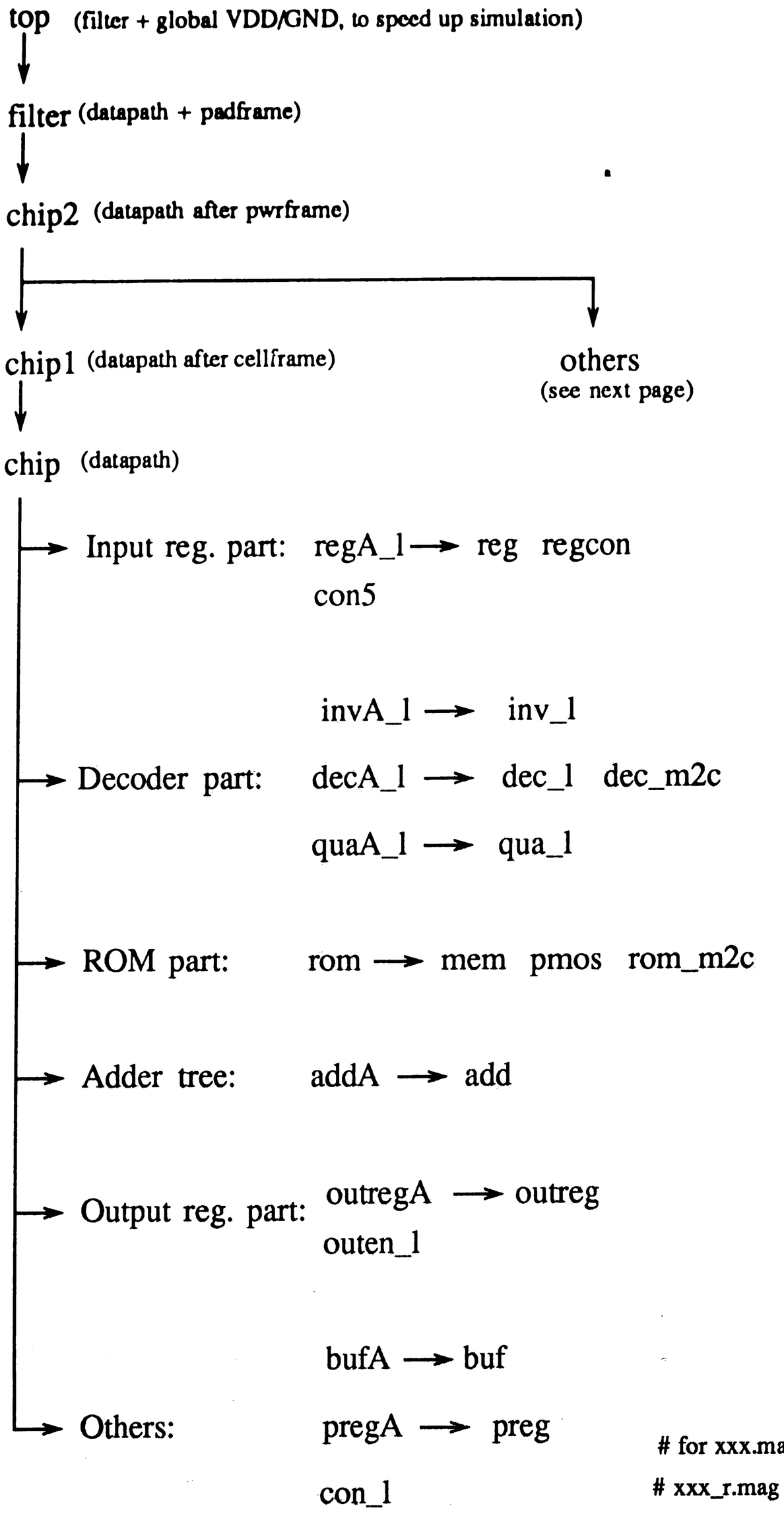
```
*** RSIM Version 6.0 ***
11990 nodes, transistors: n-channel=3534 p-channel=1956
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=X1XXXXXXXXXXXXXXXXXX1 outc=01XXXXXXXXXXXXXXXX1X0
time = 400.0ns
 [event #4543] node 544: X -> 0 @ 511.5ns
 [event #4898] node 541: X -> 0 @ 516.9ns
 [event #5071] node 544: 0 -> 1 @ 524.0ns
 [event #5770] node 544: 1 -> 0 @ 633.7ns
 [event #5786] node 541: 0 -> 1 @ 634.4ns
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01100000000000000111 outc=01011111111111111100
time = 800.0ns
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01100000000000000111 outc=01011111111111111100
time = 900.0ns
 [event #7524] node 541: 1 -> 0 @ 916.6ns
 [event #7688] node 544: 0 -> 1 @ 924.0ns
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01111111111111111100 outc=01000000000000000010
time = 1000.0ns
 [event #8371] node 544: 1 -> 0 @ 1033.7ns
 [event #8387] node 541: 0 -> 1 @ 1034.4ns
xk=0001 sx3=0001 sx2=0001 sx1=0001 sx0=0001 code0=1111111111111110
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01100000000000000111 outc=01011111111111111100
time = 1100.0ns
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
```

code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01100000000000000111 outc=01011111111111111100
time = 1200.0ns
xk=0000 sx3=0000 sx2=0000 sx1=0000 sx0=0000 code0=0111111111111111
code1=0111111111111111 code2=0111111111111111 code3=0111111111111111
outs=01100000000000000111 outc=01011111111111111100

/* timing statistics, shows path delay is 59.2ns when all pipeline stages on */
time = 1300.0ns

- 592 - 1592 161 1 631
- 582 - 1582 118 0 659
- 579 - 1579 153 0 657

APPENDIX 3: Design Files Index



- LOGO xxx.mag files

1.mag, N60.mag, 8.mag, O60.mag, 9.mag, P60.mag, A60.mag, B60.mag, C60.mag, logo.mag, D120.mag, D60.mag, E60.mag, F60.mag, G60.mag, H120.mag, R60.mag, H60.mag, S60.mag, HUANG.mag, T60.mag, copyright.mag, I60.mag, U60.mag, J60.mag, UNIVERSITY.mag, L120.mag, V60.mag, L60.mag, W120.mag, LEHIGH.mag, W60.mag, LI.mag, Y60.mag, M60.mag, a60.mag.

- standard pad layout

PadClkIn.mag, PadGnd.mag, PadGndToDP.mag, PadIn.mag, PadOut.mag, PadTri.mag, PaddVdd.mag, PadVddToDP.mag,

- padframe

  pad.info: pad listing from user.

  pad.input: pad.info after pad2frame.

  padframe.mag: pad layout file, pad.input after doing framegen.

- RSIM files

  command files: pad0.cmd, pad1.cmd, pad2.cmd

  output files: pad0.out, pad1.out, pad2.out

  0: normal mode, 1: serial output mode, 2: speed test mode

- Others

  chip_net.net: routing description file between datapath & pads.

  Top.bug: log file for top.mag rsim result.

VITA

Name: Dajen Huang

Birth date/place: May 5, 1959/ Char−I, Taiwan.

Father: Thai−Z Huang, **Mother:** Soon C. Huang

Education: received B.S. degree in electrical engineering from the National Tsing Hua University, Taiwan, in 1983, and the M.S. degree in electrical engineering from the Lehigh University, Pennsylvania, in 1989, respectively.

Work Experience: employed as integrated circuit design engineer in Texas Instruments, Taiwan, 1983−1987.